

**ĐẠI HỌC BÁCH KHOA HÀ NỘI**



# **ĐỒ ÁN TỐT NGHIỆP**

## **THIẾT KẾ HỆ THỐNG DÒ QUÉT THIẾT BỊ WIFI**

**PHẠM HỒNG DƯƠNG**

Duong.PH212741@sis.hust.edu.vn

**NGUYỄN ANH ĐỨC**

Duc.NA212769@sis.hust.edu.vn

**Ngành Kỹ thuật Điều khiển và Tự động hóa**

**Giảng viên hướng dẫn:** PGS.TS. Nguyễn Quốc Cường

Chữ ký của GVHD

**Khoa:** Tự động hóa

**Trường:** Điện - Điện tử

**Hà Nội, 06/2025**

**NHIỆM VỤ  
ĐỒ ÁN TỐT NGHIỆP**

Họ và tên sinh viên: Phạm Hồng Dương, Nguyễn Anh Đức

Khóa: K66

Trường: Điện – Điện tử

Ngành: KT ĐK & TĐH

*1. Tên đề tài*

Thiết kế hệ thống dò quét thiết bị WiFi

*2. Nội dung đề tài*

Hệ thống dò quét WiFi bao gồm các thiết bị dò quét hoạt động trên nhiều băng tần (2.4GHz, 5GHz). Các thiết bị dò quét có khả năng thu thập & trích xuất dữ liệu từ các khung bản tin Wi-Fi, lưu trữ dữ liệu, cũng như kết nối với máy chủ thông qua mạng WiFi. Khi mất kết nối với máy chủ, thiết bị sẽ tự động lưu trữ dữ liệu tại chỗ để đảm bảo không mất thông tin. Người dùng có thể điều khiển thiết bị trực tiếp tại chỗ hoặc từ xa thông qua giao diện quản trị. Phần mềm quản lý được cài đặt trên máy chủ, giúp quản lý tập trung nhiều thiết bị dò quét, xem, phân tích, xóa hoặc xuất dữ liệu thu thập được. Dưới đây là nhiệm vụ của từng cá nhân:

- Dương: Phát triển phần mềm quản lý trên máy chủ (Server)
  - Thiết kế giao diện quản trị hệ thống
  - Viết chức năng quản lý nhiều thiết bị dò quét
  - Xây dựng các chức năng xem, xóa, xuất (export) dữ liệu
  - Tích hợp các cơ chế bảo mật cho phần mềm
  - Đảm bảo phần mềm giao tiếp ổn định với các thiết bị dò quét
- Đức: Thiết kế và phát triển thiết bị dò quét WiFi
  - Viết phần mềm điều khiển thiết bị hỗ trợ quét đa băng tần (2.4GHz, 5GHz). Thu thập dữ liệu từ các khung bản tin Wi-Fi. Thu thập thông tin trạng thái kênh (Channel State Information)
  - Xây dựng chức năng truyền thông với máy chủ qua WiFi
  - Triển khai cơ chế lưu trữ dữ liệu khi mất kết nối
  - Phát triển chức năng điều khiển thiết bị từ xa và tại chỗ
  - Kiểm thử và đảm bảo thiết bị hoạt động ổn định

3. Thời gian giao đề tài: 02/2025

4. Thời gian hoàn thành: 06/2025

Ngày 16 tháng 6 năm 2025

**CÁN BỘ HƯỚNG DẪN**

**PGS.TS Nguyễn Quốc Cường**

## LỜI CẢM ƠN

Chúng em xin bày tỏ lòng biết ơn sâu sắc tới **PGS.TS Nguyễn Quốc Cường**, người đã tận tâm hướng dẫn, chỉ dạy và đưa ra những nhận xét quý báu, cụ thể trong suốt quá trình chúng em thực hiện đồ án. Sự hỗ trợ và định hướng tận tình của thầy là nguồn động lực lớn giúp chúng em hoàn thành tốt công việc của mình.

Bên cạnh đó, chúng em cũng xin gửi lời cảm ơn chân thành tới các thầy cô, anh chị và bạn bè trong Sensor Lab, những người đã luôn tạo điều kiện thuận lợi và hỗ trợ chúng em hết mình trong quá trình học tập cũng như thực hiện đồ án này.

Do kiến thức và kinh nghiệm còn hạn chế, chắc chắn trong đồ án vẫn còn những thiếu sót, sai sót. Chúng em rất mong nhận được sự góp ý, chỉ bảo từ quý thầy cô và các bạn để chúng em có thể hoàn thiện hơn trong những dự án tiếp theo.

Chúng em xin trân trọng cảm ơn!

Hà Nội, ngày 16 tháng 06 năm 2025

Sinh viên thực hiện

**Nguyễn Văn A**

## **TÓM TẮT ĐỒ ÁN**

Tóm tắt nội dung của đồ án tốt nghiệp trong khoảng tối đa 300 chữ. Phần tóm tắt cần nêu được các ý: vấn đề cần thực hiện; phương pháp thực hiện; công cụ sử dụng (phần mềm, phần cứng...); kết quả của đồ án có phù hợp với các vấn đề đã đặt ra hay không; tính thực tế của đồ án, định hướng phát triển mở rộng của đồ án (nếu có); các kiến thức và kỹ năng mà sinh viên đã đạt được.

# MỤC LỤC

<b>DANH MỤC KÝ HIỆU VÀ CHỮ VIẾT TẮT</b>	<b>i</b>
<b>DANH MỤC HÌNH VẼ</b>	<b>iii</b>
<b>DANH MỤC BẢNG BIỂU</b>	<b>iv</b>
<b>CHƯƠNG 1. GIỚI THIỆU CHUNG</b>	<b>1</b>
1.1 Bối cảnh và lý do chọn đề tài . . . . .	1
1.2 Mục tiêu nghiên cứu và thiết kế . . . . .	1
1.3 Phạm vi đồ án . . . . .	2
1.4 Phương pháp nghiên cứu . . . . .	2
1.5 Cấu trúc đồ án . . . . .	3
<b>CHƯƠNG 2. CƠ SỞ LÝ THUYẾT VÀ TỔNG QUAN CÔNG NGHỆ</b>	<b>4</b>
2.1 Tổng quan về chủ đề nghiên cứu . . . . .	4
2.2 Tổng quan về mạng không dây Wi-Fi (IEEE 802.11) . . . . .	4
2.2.1 Giới thiệu và lịch sử phát triển . . . . .	4
2.2.2 Các thành phần cơ bản & kiến trúc mạng Wi-Fi . . . . .	5
2.2.3 Cấu trúc gói tin Wi-Fi . . . . .	6
2.2.4 Các loại khung 802.11 . . . . .	8
2.2.5 CSI . . . . .	14
2.3 Netlink & nl80211 . . . . .	16
2.3.1 Netlink là gì? . . . . .	16
2.3.2 nl80211 . . . . .	17
2.4 Các giao thức khác được sử dụng . . . . .	17
2.4.1 Giao thức HTTP . . . . .	17
2.4.2 Giao thức WebSocket . . . . .	19
2.4.3 Giao thức MQTT . . . . .	21
2.5 Kiến thức Web cơ bản . . . . .	23
2.5.1 SPA (Single Page Application) . . . . .	23
2.5.2 SSR (Server Side Rendering) và CSR (Client Side Rendering) . . . . .	24
2.6 Các công nghệ liên quan . . . . .	27

2.6.1	ThingsBoard Platform . . . . .	27
2.6.2	PostgreSQL Database . . . . .	32
2.6.3	FastAPI Framework . . . . .	34
2.6.4	Next.js Language . . . . .	35
2.6.5	Crow . . . . .	35
2.6.6	MongoDB . . . . .	36
2.7	Nghiên cứu các sản phẩm/dự án tương tự . . . . .	36
2.8	Lựa chọn công nghệ phù hợp . . . . .	36
<b>CHƯƠNG 3. THIẾT KẾ HỆ THỐNG</b>		<b>37</b>
3.1	Phần mềm quản lý . . . . .	37
3.1.1	Phân tích yêu cầu hệ thống/sản phẩm . . . . .	37
3.1.2	Mô hình thiết kế tổng thể . . . . .	39
3.1.3	Thiết kế phần mềm . . . . .	40
3.2	Thiết bị dò quét . . . . .	54
3.2.1	Phân tích yêu cầu hệ thống/sản phẩm . . . . .	54
3.2.2	Mô hình thiết kế tổng thể . . . . .	55
3.2.3	Thiết kế phần mềm . . . . .	58
<b>CHƯƠNG 4. TRIỂN KHAI VÀ KIỂM THỬ</b>		<b>64</b>
4.3	Phần mềm quản lý . . . . .	64
4.3.1	Quy trình triển khai . . . . .	64
4.3.2	Thử nghiệm và đo kiểm . . . . .	81
4.3.3	Đánh giá kết quả thực tế so với thiết kế . . . . .	83
4.3.4	Vấn đề phát sinh và cách khắc phục . . . . .	83
4.4	Thiết bị dò quét . . . . .	84
4.4.1	Quy trình triển khai . . . . .	84
4.4.2	Thử nghiệm và đo kiểm . . . . .	84
4.4.3	Đánh giá kết quả thực tế so với thiết kế . . . . .	84
4.4.4	Vấn đề phát sinh và cách khắc phục . . . . .	84
<b>CHƯƠNG 5. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN</b>		<b>86</b>
5.5	Tóm tắt kết quả đạt được . . . . .	86
5.6	Những hạn chế còn tồn đọng . . . . .	86
5.7	Định hướng phát triển trong tương lai . . . . .	87

<b>KẾT LUẬN</b>	<b>88</b>
<b>TÀI LIỆU THAM KHẢO</b>	<b>89</b>
<b>PHỤ LỤC</b>	<b>91</b>
<b>A Một số phương pháp đo và hiệu chuẩn</b>	<b>91</b>



## DANH MỤC KÝ HIỆU VÀ CHỮ VIẾT TẮT

IoT	Internet of Things
AP	Access Point
STA	Station
MQTT	Message Queuing Telemetry Transport
QoS	Quality of Service
HTTP	Hypertext Transfer Protocol)
TCP	Transmission Control Protocol
URL	Uniform Resource Locator
JSON	JavaScript Object Notation
SSL	Secure Sockets Layer
TLS	Transport Layer Security
SPA	Single Page Application
AJAX	Asynchronous JavaScript and XML
SSR	Server-Side Rendering
CSR	Client-Side Rendering
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
JS	JavaScript
UI	User Interface
SEO	Search Engine Optimization
API	Application Programming Interface
REST	Representational State Transfer
RPC	Remote Procedure Call
UUID	Universally Unique Identifier
ASGI	Asynchronous Server Gateway Interface
JWT	JSON Web Token
SQL	Structured Query Language
MAC	Media Access Control
BSSID	Basic Service Set Identifier
ESSID	Extended Service Set Identifier
DB	Database
CSV	Comma-Separated Values
LocalBE	Local Backend
CRUD	Create, Read, Update, Delete
EMS	Energy Management Strategy

## DANH MỤC HÌNH VẼ

Hình 2.1.	Cấu trúc một khung 802.11 . . . . .	6
Hình 2.2.	Cấu trúc của khung quảng bá . . . . .	10
Hình 2.3.	Các loại khung điều khiển . . . . .	10
Hình 2.4.	Cấu trúc của khung dữ liệu . . . . .	12
Hình 2.5.	Quá trình giao tiếp giữa CLient và Server bằng HTTP . . . . .	17
Hình 2.6.	Quá trình giao tiếp giữa CLient và Server bằng WebSocket . . . . .	20
Hình 2.7.	Quá trình giao tiếp giữa CLient và Broker bằng MQTT . . . . .	22
Hình 2.8.	Server Side Rendering . . . . .	24
Hình 2.9.	Client Side Rendering . . . . .	26
Hình 2.10.	Kiến trúc của ThingsBoard . . . . .	28
Hình 2.11.	Mẫu Root Rule Chain . . . . .	31
Hình 2.12.	PostgreSQL Model . . . . .	33
Hình 3.1.	Kiến trúc của Phần mềm quản lý thiết bị . . . . .	39
Hình 3.2.	Trang Login . . . . .	47
Hình 3.3.	Trang Dashboard . . . . .	48
Hình 3.4.	Trang Sensor Details . . . . .	49
Hình 3.5.	Trang History . . . . .	50
Hình 3.6.	Trang User . . . . .	51
Hình 3.7.	Lưu trữ dữ liệu vào cơ sở dữ liệu ngoại . . . . .	51
Hình 3.8.	Truy vấn dữ liệu từ cơ sở dữ liệu ngoại . . . . .	52
Hình 3.9.	Cập nhật dữ liệu thời gian thực . . . . .	53
Hình 3.10.	Cập nhật thông tin thiết bị . . . . .	53
Hình 3.11.	Điều khiển thiết bị từ xa . . . . .	54
Hình 3.12.	Kiến trúc phần mềm chạy trên thiết bị . . . . .	56
Hình 3.13.	Mô hình giao tiếp giữa phần mềm ở không gian người dùng và driver của card Wi-Fi . . . . .	59
Hình 3.14.	Lưu đồ thuật toán cho việc xử lý khung quảng bá . . . . .	60
Hình 3.15.	Lưu đồ thuật toán cho việc xử lý khung dữ liệu . . . . .	60
Hình 3.16.	Mô hình giao tiếp giữa phần mềm ở không gian người dùng và driver của card Wi-Fi . . . . .	61
Hình 3.17.	Sơ đồ kiến trúc hệ thống thu thập và hiển thị dữ liệu CSI . . . . .	62
Hình 4.1.	Cài đặt và cấu hình thành công Thingsboard . . . . .	64

Hình 4.2. Giao diện Thingsboard được chạy trên localhost port 8080 . . . .	65
Hình 4.3. Cấu trúc Root Rule Chain sử dụng . . . . .	65
Hình 4.4. Luồng thêm sensor . . . . .	68
Hình 4.5. Luồng cập nhật sensor . . . . .	69
Hình 4.6. Luồng xóa sensor . . . . .	70
Hình 4.7. Luồng lấy danh sách card mạng của sensor . . . . .	71
Hình 4.8. Luồng thiết lập trạng thái card từ người dùng . . . . .	72
Hình 4.9. Luồng bật tắt nhận hiển thị kết quả quét từ người dùng . . . . .	72
Hình 4.10. Luồng truy vấn danh sách AP . . . . .	73
Hình 4.11. Luồng truy vấn danh sách STA . . . . .	73
Hình 4.12. Luồng truy vấn STA của 1 AP tại 1 thời điểm . . . . .	74
Hình 4.13. Luồng truy vấn danh sách Notification . . . . .	74
Hình 4.14. Luồng truy vấn lịch sử AP . . . . .	75
Hình 4.15. Luồng truy vấn lịch sử STA . . . . .	75
Hình 4.16. Luồng cập nhật kết quả quét realtime trên giao diện hiển thị . . .	76
Hình 4.17. Luồng cập nhật trạng thái thiết bị realtime trên giao diện hiển thị	76
Hình 4.18. Luồng cập nhật thông báo realtime trên giao diện hiển thị . . . .	77
Hình 4.19. Luồng cập nhật thông tin thiết bị vào database ngoại từ người dùng	77
Hình 4.20. Luồng lưu kết quả quét vào database ngoại . . . . .	78
Hình 4.21. Luồng đăng nhập giao diện người dùng . . . . .	79
Hình 4.22. Luồng cập nhật thông tin người dùng . . . . .	80
Hình 4.23. Cấu hình PostgreSQL . . . . .	81
Hình 4.24. Các bảng trong cơ sở dữ liệu ngoài . . . . .	81
Hình 4.25. Đo thời gian phản hồi API xác thực người dùng thất bại . . . . .	82
Hình 4.26. Đo thời gian phản hồi API xác thực người dùng thành công . . .	83

## DANH MỤC BẢNG BIỂU

Bảng 0.1.	Bảng cập nhật báo cáo. . . . .	v
Bảng 0.2.	Bảng kế hoạch dự án. . . . .	vi
Bảng 0.3.	Biên bản cuộc họp. . . . .	vii
Bảng 2.1.	Các loại khung quản lý . . . . .	8
Bảng 2.2.	Ý nghĩa của trường địa chỉ khi ToDS = 0, FromDS = 0 . . . . .	13
Bảng 2.3.	Ý nghĩa của trường địa chỉ khi ToDS = 1, FromDS = 0 . . . . .	13
Bảng 2.4.	Ý nghĩa của trường địa chỉ khi ToDS = 0, FromDS = 1 . . . . .	14
Bảng 2.5.	Ý nghĩa của trường địa chỉ khi ToDS = 1, FromDS = 1 . . . . .	14
Bảng 3.1.	Yêu cầu chức năng của Hệ thống quản lý thiết bị . . . . .	38
Bảng 3.2.	Yêu cầu chức năng của phần mềm chạy trên thiết bị . . . . .	55
Bảng 4.1.	Quy tắc xử lý dữ liệu trong Rule Chain . . . . .	66

**Bảng cập nhật báo cáo**

*Bảng 0.1. Bảng cập nhật báo cáo.*

Ngày	Nội dung báo cáo	Sửa đổi / ghi chú
01/10	Chương 1: Giới thiệu	Cập nhật mục tiêu nghiên cứu và phạm vi
08/10	Chương 3: Phương pháp	Thêm mô tả chi tiết về quy trình thực hiện
15/10	Chương 4: Kết quả thực nghiệm	Cập nhật kết quả phân tích dữ liệu mới
22/10	Toàn bộ báo cáo	Chỉnh sửa ngôn ngữ, định dạng

**Kế hoạch thực hiện**

*Bảng 0.2. Bảng kế hoạch dự án.*

Tuần	Nhiệm vụ	Yêu cầu cần đạt	Trạng thái
24	Nghiên cứu tài liệu liên quan	Tóm tắt tài liệu, xác định phương pháp	Hoàn thành
25	Thiết kế sơ đồ khối hệ thống	Bản thiết kế sơ đồ khối	Đang thực hiện

**Biên bản cuộc họp**

*Bảng 0.3. Biên bản cuộc họp.*

Ngày	Nội dung	Quyết định	Nhiệm vụ tiếp theo
01/10	Thảo luận về thiết kế hệ thống	Sẽ thử với phương pháp A trước	viết báo cáo chương 2

# CHƯƠNG 1. GIỚI THIỆU CHUNG

## 1.1 Bối cảnh và lý do chọn đề tài

Trong kỷ nguyên số, mạng không dây (WiFi) đã trở thành một hạ tầng thiết yếu, không thể thiếu trong mọi hoạt động từ cá nhân, doanh nghiệp đến các tổ chức chính phủ. Sự phổ biến của các chuẩn WiFi mới như WiFi 6 và WiFi 6E đã mở rộng băng tần hoạt động lên 2.4 GHz, 5 GHz và 6 GHz, mang lại tốc độ cao hơn nhưng cũng đặt ra những thách thức mới về quản lý, giám sát và bảo mật.

Việc quản lý một hệ thống mạng WiFi phức tạp đòi hỏi các công cụ chuyên dụng có khả năng giám sát liên tục, phát hiện các thiết bị trái phép (rogue access points), phân tích hiệu suất mạng và xác định các lỗ hổng bảo mật. Các công cụ hiện có thường tồn tại một số hạn chế như: chi phí cao, chỉ hỗ trợ các băng tần cũ, hoặc thiếu khả năng quản lý tập trung nhiều địa điểm cùng lúc. Đặc biệt, việc thiếu một hệ thống có khả năng hoạt động độc lập, lưu trữ dữ liệu tạm thời khi mất kết nối và được quản lý từ xa là một rào cản lớn cho việc triển khai linh hoạt.

Từ những nhu cầu và thách thức thực tiễn đó, chúng em đề xuất đề tài **"Hệ thống dò quét thiết bị WiFi"** nhằm xây dựng một giải pháp toàn diện để giải quyết các vấn đề trên. Hệ thống này không chỉ cung cấp khả năng dò quét đa băng tần mà còn cho phép quản lý tập trung, linh hoạt và hiệu quả, đáp ứng nhu cầu giám sát an ninh mạng không dây trong môi trường hiện đại.

## 1.2 Mục tiêu nghiên cứu và thiết kế

Mục tiêu chính của đề án là nghiên cứu, thiết kế và xây dựng thành công một hệ thống dò quét WiFi hoàn chỉnh, bao gồm hai thành phần chính: thiết bị dò quét và phần mềm quản lý trung tâm. Cụ thể, các mục tiêu cần đạt được bao gồm:

- Đối với Thiết bị dò quét:
  - Thiết kế và chế tạo một thiết bị nhỏ gọn dựa trên máy tính nhúng.
  - Tích hợp khả năng dò quét trên cả ba băng tần: 2.4 GHz, 5 GHz và 6 GHz.
  - Đảm bảo thiết bị có thể truyền dữ liệu thu thập được về Server quản lý thông qua kết nối WiFi.
  - Xây dựng cơ chế lưu trữ dữ liệu đệm khi thiết bị mất kết nối với Server và tự động đồng bộ lại khi có kết nối.
  - Cho phép điều khiển thiết bị theo hai chế độ: Local (trực tiếp) và Remote (từ xa qua Server).
- Đối với Phần mềm quản lý:



- Xây dựng một Server trung tâm có khả năng quản lý đồng thời nhiều thiết bị dò quét.
- Phát triển giao diện quản lý cho phép người dùng xem, lọc, xóa và xuất dữ liệu do các thiết bị gửi về.
- Triển khai các cơ chế bảo mật cơ bản như xác thực người dùng và mã hóa dữ liệu truyền nhận.
- Sản phẩm cuối cùng:
  - Chế tạo thành công ít nhất 01 thiết bị dò quét hoạt động ổn định.
  - Triển khai Server cài đặt phần mềm quản lý và kết nối thành công với thiết bị dò quét.

### 1.3 Phạm vi đề án

Đề án tập trung vào việc thiết kế, tích hợp và phát triển phần mềm cho hệ thống, với các giới hạn cụ thể như sau:

- Về phần cứng: Đề án không đi sâu vào thiết kế vi mạch mà tập trung vào việc lựa chọn và tích hợp các linh kiện có sẵn trên thị trường như máy tính nhúng (Raspberry Pi), các module WiFi tương thích và các linh kiện phụ trợ để tạo thành một thiết bị hoàn chỉnh.
- Về phần mềm:
  - Phần mềm trên thiết bị: Xây dựng trang Web Local để điều khiển module WiFi, thu thập dữ liệu, xử lý và giao tiếp với Server.
  - Phần mềm quản lý: Xây dựng trang Web Remote với các chức năng quản lý cốt lõi để quản lý nhiều thiết bị.
- Về triển khai: Hệ thống sẽ được triển khai và kiểm thử trong môi trường phòng thí nghiệm hoặc một mạng văn phòng nhỏ để đánh giá hiệu quả hoạt động.

### 1.4 Phương pháp nghiên cứu

Để hoàn thành các mục tiêu đề ra, nhóm sẽ áp dụng kết hợp các phương pháp sau:

- Nghiên cứu lý thuyết: Tìm hiểu sâu về các chuẩn giao thức WiFi (802.11a/b/g/n/ac/ax), kỹ thuật dò quét mạng (packet sniffing), kiến trúc hệ thống nhúng, và mô hình Client-Server.
- Nghiên cứu so sánh: Khảo sát các giải pháp dò quét WiFi thương mại và mã nguồn mở (như Kismet, Wireshark) để học hỏi về tính năng và lựa chọn phương án thiết kế tối ưu.

- Phương pháp thực nghiệm và phát triển theo mẫu: Xây dựng hệ thống theo từng giai đoạn. Thiết bị phần cứng và phần mềm quản lý sẽ được phát triển song song, sau đó tiến hành tích hợp và kiểm thử liên tục.
- Phân công công việc:
  - Phạm Hồng Dương: Chịu trách nhiệm chính về phần mềm quản lý trên Server.
  - Nguyễn Anh Đức: Chịu trách nhiệm chính về thiết bị dò quét.

## 1.5 Cấu trúc đồ án

Nội dung của đồ án được trình bày trong 5 chương:

- Chương 1: Giới thiệu chung – Trình bày tổng quan về đề tài, lý do, mục tiêu, phạm vi và phương pháp thực hiện.
- Chương 2: Cơ sở lý thuyết và tổng quan công nghệ – Phân tích các kiến thức nền tảng về mạng WiFi, kỹ thuật dò quét, hệ thống nhúng, và các công nghệ được sử dụng.
- Chương 3: Thiết kế hệ thống – Mô tả chi tiết kiến trúc của hệ thống, bao gồm thiết kế phần cứng, phần mềm nhúng cho thiết bị dò quét, và kiến trúc phần mềm cho Server quản lý.
- Chương 4: Triển khai và kiểm thử – Trình bày quá trình hiện thực hóa sản phẩm, kết quả triển khai và đánh giá các chức năng của hệ thống.
- Chương 5: Kết luận và hướng phát triển – Tóm tắt các kết quả đạt được, so sánh với mục tiêu ban đầu và đề xuất các hướng phát triển trong tương lai.

### Kết luận chương

Chương 1 đã phác thảo một cách toàn diện về đề tài "Hệ thống dò quét thiết bị WiFi". Các mục tiêu, phạm vi và phương pháp thực hiện đã được xác định rõ ràng, tạo nền tảng vững chắc cho việc triển khai các chương tiếp theo. Việc hiểu rõ bối cảnh và yêu cầu kỹ thuật là yếu tố then chốt để đảm bảo sản phẩm cuối cùng đáp ứng đúng nhu cầu thực tiễn.

## CHƯƠNG 2. CƠ SỞ LÝ THUYẾT VÀ TỔNG QUAN CÔNG NGHỆ

### 2.1 Tổng quan về chủ đề nghiên cứu

### 2.2 Tổng quan về mạng không dây Wi-Fi (IEEE 802.11)

#### 2.2.1 Giới thiệu và lịch sử phát triển

##### 2.2.1.1 Khái niệm về Mạng không dây (WLAN)

Mạng cục bộ không dây (WLAN - Wireless Local Area Network) là một loại mạng máy tính cho phép các thiết bị kết nối và giao tiếp với nhau mà không cần sử dụng dây cáp vật lý như mạng Ethernet truyền thống. Thay vào đó, WLAN sử dụng sóng vô tuyến (Radio Frequency - RF) để truyền và nhận dữ liệu. Ưu điểm chính của WLAN là sự linh hoạt, tính di động cao, cho phép người dùng kết nối mạng ở bất cứ đâu trong phạm vi phủ sóng mà không bị ràng buộc bởi dây cáp, đồng thời giúp việc lắp đặt và mở rộng mạng trở nên dễ dàng hơn.

##### 2.2.1.2 Chuẩn IEEE 802.11 và Wi-Fi

Để đảm bảo tính tương thích và khả năng tương tác giữa các thiết bị mạng không dây từ nhiều nhà sản xuất khác nhau, Viện Kỹ sư Điện và Điện tử (IEEE - Institute of Electrical and Electronics Engineers) đã phát triển và ban hành một bộ tiêu chuẩn kỹ thuật quốc tế cho WLAN, được biết đến với tên gọi IEEE 802.11. Bộ chuẩn này định nghĩa các thông số kỹ thuật cho cả lớp vật lý (PHY - Physical Layer) và lớp điều khiển truy nhập đường truyền (MAC - Medium Access Control) của mạng không dây.

Song song với chuẩn kỹ thuật IEEE 802.11, thuật ngữ "Wi-Fi" thường được sử dụng rộng rãi hơn trong đời sống. "Wi-Fi" thực chất là tên thương hiệu được đăng ký bởi tổ chức Wi-Fi Alliance. Wi-Fi Alliance là một hiệp hội các công ty công nghệ nhằm thúc đẩy công nghệ WLAN và chứng nhận các sản phẩm tuân thủ các tiêu chuẩn IEEE 802.11. Một sản phẩm được chứng nhận "Wi-Fi CERTIFIED" đảm bảo rằng nó đã được kiểm tra về khả năng tương thích với các thiết bị Wi-Fi khác. Vì vậy, mặc dù về mặt kỹ thuật "IEEE 802.11" là tên chuẩn, "Wi-Fi" là tên gọi phổ thông và thương hiệu đảm bảo chất lượng cho công nghệ mạng không dây dựa trên chuẩn này.

##### 2.2.1.3 Lịch sử phát triển

Công nghệ Wi-Fi đã trải qua nhiều thế hệ phát triển, mỗi thế hệ mang đến những cải tiến đáng kể về tốc độ, hiệu quả sử dụng băng tần và các tính năng mới. Cụ thể:

- IEEE 802.11 (1997): Phiên bản gốc, đặt nền móng cho mạng không dây. Hoạt động ở băng tần 2.4 GHz, sử dụng các kỹ thuật trải phổ như DSSS (Direct-Sequence Spread Spectrum) và FHSS (Frequency-Hopping Spread Spectrum). Tốc độ rất hạn chế, chỉ đạt 1-2 Mbps. Ngày nay chuẩn này đã hoàn toàn lỗi thời.
- IEEE 802.11b (1999): Tạo ra bước đột phá đầu tiên về tốc độ và tính phổ biến.

Tiếp tục sử dụng băng tần 2.4 GHz nhưng cải tiến kỹ thuật DSSS (sử dụng CCK - Complementary Code Keying). Tốc độ tối đa tăng lên 11 Mbps. Đây là chuẩn giúp Wi-Fi bắt đầu được áp dụng rộng rãi. Tuy nhiên, băng tần 2.4 GHz bắt đầu trở nên đông đúc và dễ bị nhiễu từ các thiết bị khác (lò vi sóng, điện thoại không dây...).

- IEEE 802.11a (1999): Ra mắt cùng thời điểm với 802.11b nhưng hoạt động ở băng tần 5 GHz ít nhiễu hơn. Sử dụng kỹ thuật điều chế hiệu quả hơn là OFDM (Orthogonal Frequency-Division Multiplexing). Tốc độ tối đa đạt 54 Mbps. Nhược điểm là tầm phủ sóng ngắn hơn và chi phí thiết bị ban đầu cao hơn 802.11b.
- IEEE 802.11g (2003): Kết hợp ưu điểm của hai chuẩn trước. Hoạt động ở băng tần 2.4 GHz (đảm bảo tương thích ngược với 802.11b) nhưng sử dụng điều chế OFDM (như 802.11a) để đạt tốc độ tối đa 54 Mbps. Chuẩn này nhanh chóng thay thế 802.11b và trở nên rất phổ biến.
- IEEE 802.11n (2009): Một bước nhảy vọt về hiệu năng. Lần đầu tiên giới thiệu công nghệ MIMO (Multiple-Input Multiple-Output), sử dụng nhiều anten để truyền và nhận dữ liệu đồng thời, tăng đáng kể tốc độ và độ ổn định. Hoạt động trên cả hai băng tần 2.4 GHz và 5 GHz. Hỗ trợ ghép kênh (Channel Bonding) tạo kênh rộng 40 MHz. Tốc độ tối đa lý thuyết có thể lên tới 600 Mbps (tùy thuộc vào số luồng MIMO và độ rộng kênh). Wi-Fi Alliance bắt đầu sử dụng tên gọi thế hệ, gọi đây là Wi-Fi 4.
- IEEE 802.11ac (2013 - Wi-Fi 5): Tập trung cải thiện hiệu năng trên băng tần 5 GHz (ít nhiễu hơn). Mở rộng độ rộng kênh lên 80 MHz và tùy chọn 160 MHz. Giới thiệu MU-MIMO (Multi-User MIMO) cho chiều xuống (Downlink), cho phép AP truyền dữ liệu đến nhiều thiết bị cùng lúc. Sử dụng điều chế phức tạp hơn (256-QAM). Tốc độ tối đa lý thuyết đạt mức hàng Gigabit mỗi giây (ví dụ: 1.3 Gbps hoặc cao hơn). Được gọi là Wi-Fi 5.
- IEEE 802.11ax (2019/2020): Không chỉ tập trung vào tốc độ tối đa mà còn cải thiện đáng kể hiệu quả sử dụng mạng, đặc biệt trong môi trường mật độ thiết bị cao. Hoạt động trên cả 2.4 GHz và 5 GHz. Công nghệ chủ chốt là OFDMA (Orthogonal Frequency-Division Multiple Access), cho phép chia nhỏ kênh tần số để phục vụ nhiều thiết bị cùng lúc trên cả chiều lên và xuống, giảm độ trễ và tăng hiệu quả.

### ***2.2.2 Các thành phần cơ bản & kiến trúc mạng Wi-Fi***

Các mạng Wi-Fi hiện nay thường sẽ có các thiết bị như sau:

- Access Point (AP - Điểm Truy cập): Là thiết bị phần cứng đóng vai trò trung tâm trong kiến trúc mạng cơ sở hạ tầng (Infrastructure Mode). Nó tạo ra một mạng không dây cục bộ (WLAN) mà các thiết bị khác có thể kết nối vào.
- Station (STA): Là bất kỳ thiết bị nào có khả năng kết nối mạng không dây (thông qua card mạng Wi-Fi tích hợp hoặc gắn ngoài) và tham gia vào một mạng WLAN bằng cách kết nối tới một Access Point.
- Repeater / Range Extender (Bộ lặp / Bộ mở rộng sóng): Là thiết bị mạng được sử dụng để mở rộng phạm vi phủ sóng của một mạng WLAN hiện có. Nó hữu ích ở những khu vực mà tín hiệu Wi-Fi từ AP chính yếu hoặc không tới được.

**2.2.2.1 Chế độ Ad-hoc (IBSS - Independent Basic Service Set)** Các thiết bị kết nối trực tiếp với nhau không cần Access Point (AP). Ít phổ biến hơn.

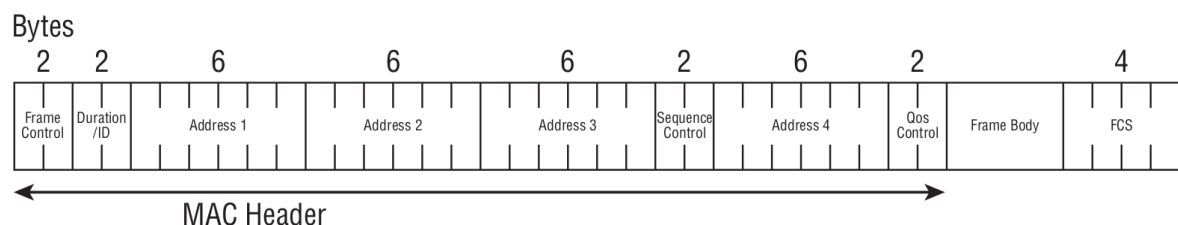
**2.2.2.2 Chế độ cơ sở hạ tầng (Infrastructure Mode)** Mô hình phổ biến nhất, kiến trúc có thể là BSS hoặc nhiều BSS được kết nối với nhau (ESS). Cụ thể như sau:

- BSS (Basic Service Set): Gồm một AP và các trạm (STA - Station) kết nối với AP đó. Định danh bởi BSSID (thường là địa chỉ MAC của AP)
- ESS (Extended Service Set): Gồm nhiều BSS được kết nối với nhau thông qua hệ thống phân phối (Distribution System - DS, thường là mạng Ethernet có dây), chia sẻ cùng một SSID (Service Set Identifier - tên mạng).

### 2.2.3 Cấu trúc gói tin Wi-Fi

Để thực hiện việc giám sát và phân tích mạng Wi-Fi hiệu quả, việc hiểu rõ cấu trúc của gói tin dữ liệu được truyền qua sóng vô tuyến là cực kỳ quan trọng. Theo chuẩn kỹ thuật IEEE 802.11, gói tin dữ liệu này được gọi là khung (Frame).

Mỗi khung, bất kể loại nào (Quản lý, Điều khiển, Dữ liệu), đều có một cấu trúc cơ bản gồm các trường chính sau đây trong phần header và trailer.



Hình 2.1. Cấu trúc một khung 802.11

Ta có thể thấy trên hình 2.1. có các trường trong khung 802.11 như sau:

#### 2.2.3.1 Frame Control (2 bytes)

Trường quan trọng xác định phiên bản giao thức, loại khung (management, control, data), kiểu khung cụ thể (beacon, probe request, ACK, data...), và chứa các cờ điều khiển khác nhau.

#### **2.2.3.2 Duration/ID (2 bytes)**

Trường này có hai ý nghĩa, tùy vào loại khung. Cụ thể như sau:

- Trong hầu hết các khung: Chỉ định khoảng thời gian (tính bằng micro giây -  $\mu s$ ) mà môi trường truyền sẽ bị chiếm dụng bởi khung hiện tại và các khung liên quan (như ACK). Giá trị này được các trạm khác sử dụng để cập nhật Vector Phân bổ Mạng (NAV - Network Allocation Vector) của chúng, nhằm tránh xung đột (cơ chế CSMA/CA).
- Trong khung PS-Poll (Power Save Poll): Chứa Association ID (AID) của trạm đang gửi khung này để yêu cầu dữ liệu đệm từ AP.

#### **2.2.3.3 Address 1,2,3 (Trường Địa chỉ - 4 trường, mỗi trường 6 bytes)**

Chứa các địa chỉ MAC liên quan đến khung. Ý nghĩa cụ thể của từng trường (Address1 đến Address4) phụ thuộc vào hướng đi của khung (giữa STA, AP, và Hệ thống Phân phối - DS), được xác định bởi các cờ trong trường Frame Control.

#### **2.2.3.4 Sequence Control (2 bytes)**

Dùng để đánh số thứ tự các khung và hỗ trợ việc phân mảnh/tái hợp các khung lớn.

#### **2.2.3.5 Address 4 (6 bytes)**

Trường địa chỉ thứ 4, chỉ được sử dụng trong các kịch bản cụ thể như Wireless Distribution System (WDS) giữa các AP. Trong hầu hết các trường hợp thông thường, trường này không có mặt hoặc không được sử dụng.

#### **2.2.3.6 Frame Body (Thân khung - Kích thước thay đổi, 0-2312 bytes)**

Trường này chứa phần dữ liệu thực tế của khung. Tùy theo loại khung cụ thể mà dữ liệu khác nhau. Ví dụ:

- Đối với khung Quản lý: Chứa các thông tin quản lý (như tham số mạng trong Beacon).
- Đối với khung Dữ liệu: Chứa dữ liệu từ lớp cao hơn (thường là gói tin IP), có thể được mã hóa.
- Đối với khung Điều khiển: Không có.

#### **2.2.3.7 FCS (Frame Check Sequence - Chuỗi kiểm tra khung, 4 bytes)**

Nằm ở cuối khung (trailer). Chứa giá trị kiểm tra lỗi CRC (Cyclic Redundancy Check) 32-bit, được tính toán trên toàn bộ các trường từ Frame Control đến Frame Body. Bên nhận sẽ tính toán lại FCS và so sánh, nếu khớp nghĩa là khung được truyền đến nguyên vẹn, không bị lỗi bit.

## 2.2.4 Các loại khung 802.11

### 2.2.4.1 Khung quản lý

Mỗi khung 802.11 đều có một trường gọi là Frame Control. Bên trong trường này, có hai trường con là Type (2 bit) và Subtype (4 bit) dùng để xác định chức năng của khung. Các khung quản lý đều có giá trị trường Type giống nhau là 00.

Khung quản lý thường là thành phần quan trọng của mạng Wi-Fi. Chúng cho phép các thiết bị không dây hình thành kết nối với nhau và, đúng như tên gọi, “quản lý” các kết nối đó.

*Bảng 2.1. Các loại khung quản lý*

<b>Giá trị trường Subtype</b>	<b>Loại khung</b>
0000	Association Request
0001	Association Response
0010	Reassociation Request
0011	Reassociation Response
0100	Probe Request
0101	Probe Response
1000	Beacon
1001	Announcement traffic indication message (ATIM)
1010	Disassociation
1011	Authentication
1100	Deauthentication
1101	Action
1110	Action No Ack

- Chức năng của từng loại khung quản lý như sau
  - Association Request: Trạm (station) gửi yêu cầu đến điểm truy cập (AP) để “gia nhập” (associate) vào mạng
  - Association Response: AP phản hồi chấp nhận hoặc từ chối yêu cầu gia nhập của trạm
  - Reassociation Request: Trạm đang liên kết rồi gửi yêu cầu chuyển đổi (“chuyển vùng”) sang AP khác trong cùng ESS
  - Reassociation Response: AP mới trả lời chấp nhận hoặc từ chối việc chuyển vùng của trạm
  - Probe Request: Trạm tìm kiếm AP hoặc SSID cụ thể bằng cách phát tín hiệu dò (active scan)

- Probe Response: AP phản hồi các thông tin (SSID, khả năng hỗ trợ...) cho trạm dò.
- Beacon: AP phát định kỳ để quảng bá SSID, các tính năng, thông số mạng
- ATIM: Trong mạng IBSS (ad-hoc), khung dùng để báo có dữ liệu chờ gửi sau giai đoạn ngủ
- Disassociation: Một bên thông báo kết thúc kết nối (association) vẫn còn xác thực, nhưng không còn trao đổi dữ liệu
- Authentication: Khung được trạm gửi để xin xác thực với AP trước khi thực sự nó gia nhập vào mạng
- Deauthentication: Khung kết thúc hoàn toàn phiên xác thực; trạm và AP phải quay lại bước xác thực nếu muốn tiếp tục kết nối
- Action: Khung đa năng cho các thao tác nâng cao (block-ack, đo đặc kênh, quản lý năng lượng, fast-roaming...)
- Action No Ack: Giống khung Action nhưng không yêu cầu khung ACK, dùng khi tốc độ cao hoặc trong môi trường đặc biệt

- Khung quảng bá (Beacon)

Để kết nối mạng không dây, các thiết bị như máy tính xách tay hay điện thoại sử dụng phương pháp quét thụ động. Chúng sẽ lần lượt lắng nghe trên từng kênh sóng để phát hiện tín hiệu từ các điểm truy cập (AP). Để theo dõi các điểm truy cập, đồ án cũng sử dụng phương pháp tương tự, đó là thụ động lắng nghe những khung quảng bá.

Các điểm truy cập sẽ liên tục phát ra các gói tin đặc biệt gọi là khung quảng bá. Các khung quảng bá này chứa thông tin quan trọng về mạng, chẳng hạn như tên mạng, các chuẩn được hỗ trợ, và các thông số kỹ thuật khác.

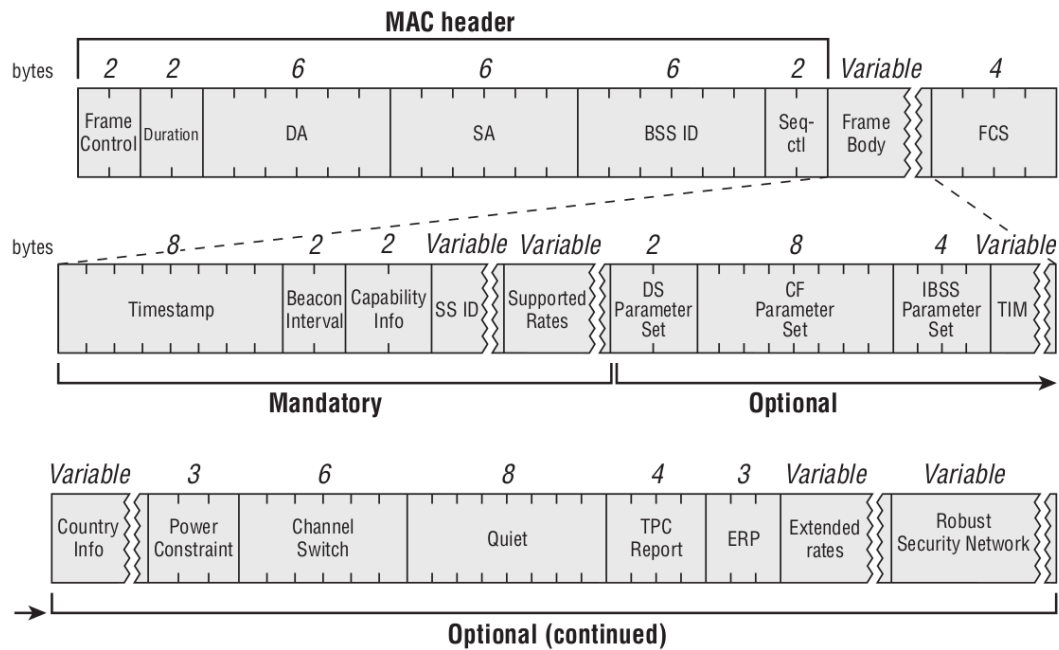
Mục đích chính của khung quảng bá như sau

- Quảng bá SSID: Cho các thiết bị đang dò thấy tên mạng (SSID), cho phép chúng biết có mạng nào xung quanh
- Công bố tham số mạng: Kênh, chế độ bảo mật, tốc độ hỗ trợ...
- Đồng bộ hoá thời gian: Các trạm dùng trường Timestamp để điều chỉnh bộ đếm nội bộ, đảm bảo nó và các trạm khác đang có cùng một mốc tham chiếu thời gian

#### 2.2.4.2 Khung điều khiển

Vai trò chính của khung điều khiển là hỗ trợ việc truyền các khung dữ liệu và khung quản lý một cách đáng tin cậy. Chúng có cấu trúc rất đơn giản, không có phần





Hình 2.2. Cấu trúc của khung quảng bá

thân (frame body) mà chỉ bao gồm phần đầu (header) và phần cuối (trailer) của lớp MAC.

Các khung điều khiển đều có giá trị trường Type giống nhau là 01.

Type value b3 b2	Type description	Subtype value b7 b6 b5 b4	Subtype description
01	Control	0000–0110	Reserved
01	Control	0111	Control wrapper
01	Control	1000	Block ack request (BlockAckReq)
01	Control	1001	Block ack (BlockAck)
01	Control	1010	PS-Poll
01	Control	1011	RTS
01	Control	1100	CTS
01	Control	1101	ACK
01	Control	1110	CF-End
01	Control	1111	CF-End and CF-Ack

Hình 2.3. Các loại khung điều khiển

- Chức năng của từng loại khung điều khiển như sau

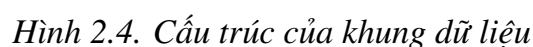
- Control Wrapper: Được sử dụng trong các chuẩn Wi-Fi mới hơn (như 802.11n/ac/ax) để "gói" các khung điều khiển cũ hơn. Điều này cho phép các thiết bị cũ vẫn hoạt động trong một mạng có các thiết bị hiệu suất cao (High Throughput - HT) hoặc cao hơn
- BlockAckReq (Block Acknowledgement Request): Yêu cầu xác nhận cho một khối các khung dữ liệu thay vì từng khung một. Bên gửi sẽ gửi BlockAckReq để thiết lập một phiên báo nhận theo khối, giúp tăng hiệu suất truyền dữ liệu
- BlockAck (Block Acknowledgement): Là khung trả lời cho BlockAckReq. Nó chứa một bản đồ bit (bitmap) cho biết khung nào trong khối đã được nhận thành công và khung nào bị mất, cho phép bên gửi chỉ cần truyền lại những khung bị lỗi thay vì cả khối
- PS-Poll (Power Save-Poll): Được một thiết bị (trạm) đang ở chế độ tiết kiệm năng lượng gửi đến Điểm truy cập (AP). Nó dùng để yêu cầu AP gửi bất kỳ dữ liệu nào đang được lưu trữ đệm cho nó
- RTS (Request to Send): Một thiết bị gửi khung RTS để "đặt chỗ" kênh truyền sóng. Nó báo cho tất cả các thiết bị khác trong vùng phủ sóng rằng nó muốn bắt đầu truyền dữ liệu và yêu cầu chúng im lặng trong một khoảng thời gian nhất định. Đây là một cơ chế để tránh xung đột dữ liệu, đặc biệt hữu ích trong các mạng đông đúc hoặc khi có các thiết bị ở xa không "nghe" thấy nhau (vấn đề trạm ẩn)
- CTS (Clear to Send): Là khung trả lời cho RTS. Điểm truy cập (AP) gửi lại CTS cho thiết bị yêu cầu và cho tất cả các thiết bị khác trong vùng phủ sóng. Nó xác nhận rằng kênh đã sẵn sàng và ra lệnh cho các thiết bị khác không được truyền tín hiệu trong khoảng thời gian được chỉ định
- ACK (Acknowledgement): Là một khung xác nhận. Thiết bị nhận sẽ gửi lại khung ACK cho thiết bị gửi để báo rằng nó đã nhận thành công một khung dữ liệu (hoặc quản lý) trước đó. Nếu bên gửi không nhận được ACK, nó sẽ hiểu rằng gói tin đã bị mất và sẽ truyền lại
- CF-End (Contention-Free End): Báo hiệu sự kết thúc của "Giai đoạn không xung đột" (Contention-Free Period - CFP) trong các mạng có sử dụng Point Coordination Function (PCF), một cơ chế truy cập kênh ít phổ biến hơn
- CF-End & CF-Ack (Contention-Free End & Acknowledgement): Kết hợp chức năng của CF-End và ACK. Nó vừa báo hiệu kết thúc giai đoạn không xung đột, vừa xác nhận đã nhận thành công khung dữ liệu cuối cùng

### 2.2.4.3 Khung dữ liệu

- Cấu trúc cơ bản
  - MAC Header: Chứa các thông tin điều khiển phục vụ cho lớp liên kết dữ liệu
  - Frame Body: Chứa dữ liệu của các lớp trên, thường là một gói tin IP. Phần này có thể được mã hóa để bảo mật
  - FCS (Frame Check Sequence): Một trường 4 byte ở cuối cùng, chứa giá trị kiểm tra lỗi (CRC). Bên nhận sẽ tính toán lại và so sánh để đảm bảo khung không bị lỗi trong quá trình truyền

Khung dữ liệu có tới 15 loại (Subtype) khác nhau để tối ưu hóa cho nhiều tình huống sử dụng khác nhau. Các loại khung này được tạo ra bằng cách thay đổi giá trị của 4 bit trong trường Subtype. Mỗi bit có một ý nghĩa riêng:

- Bit 7 (QoS): Khi bật, khung này trở thành khung QoS Data, khiến khung này được ưu tiên, phù hợp cho các ứng dụng thời gian thực như xem video. Đây là loại khung dữ liệu phổ biến nhất trong các mạng Wi-Fi hiện đại
- Bit 6 (Null): Khi bật, khung này là khung Null (không chứa dữ liệu trong phần thân). Nó được dùng cho các mục đích quản lý như để trạm báo cho AP biết nó đang chuyển sang chế độ tiết kiệm năng lượng
- Bit 5 (+CF-Poll) và Bit 4 (+CF-Ack): Thêm các chức năng thăm dò (poll) và báo nhận (ack) trong chế độ hoạt động không xung đột (Contention-Free), ít phổ biến hơn hiện nay



Bốn địa chỉ trong phần MAC header của khung dữ liệu có ý nghĩa khác nhau và phụ thuộc vào cặp bit "To DS" và "From DS" trong trường Frame Control

- To DS: Bit này bằng 1 nếu khung dữ liệu đang được gửi tới Hệ thống phân phối (Distribution System - DS). Hiểu đơn giản, DS chính là mạng có dây mà Access Point (AP) được kết nối vào
- From DS: Bit này bằng 1 nếu khung dữ liệu được gửi từ Hệ thống phân phối

Tổ hợp của 2 bit này sẽ quyết định ý nghĩa của từng trường địa chỉ. Có 4 trường hợp xảy ra:

1. Trường hợp 1: To DS = 0, From DS = 0

Đây là kịch bản giao tiếp trong mạng Ad-hoc (IBSS), nơi các trạm (STA) giao tiếp trực tiếp với nhau mà không cần AP. Bảng 2.2. cho biết ý nghĩa của từng trường địa chỉ

*Bảng 2.2. Ý nghĩa của trường địa chỉ khi ToDS = 0, FromDS = 0*

Trường	Vai trò	Diễn giải
Address 1	RA = DA	Địa chỉ MAC của trạm nhận
Address 2	TA = SA	Địa chỉ MAC của trạm gửi
Address 3	BSSID	Mã định danh của mạng Ad-hoc
Address 4	Không dùng	

2. Trường hợp 2: To DS = 1, From DS = 0

Một trạm (STA) gửi dữ liệu lên Access Point (AP) để đi ra ngoài (ví dụ: truy cập Internet). Bảng 2.3. cho biết ý nghĩa của từng trường địa chỉ

*Bảng 2.3. Ý nghĩa của trường địa chỉ khi ToDS = 1, FromDS = 0*

Trường	Vai trò	Diễn giải
Address 1	RA = BSSID	Địa chỉ MAC của AP
Address 2	TA = SA	Địa chỉ MAC của trạm gửi
Address 3	DA	Địa chỉ MAC của đích cuối cùng (thường là địa chỉ của router trong mạng dây)
Address 4	Không dùng	

3. Trường hợp 3: To DS = 0, From DS = 1

Điểm truy cập (AP) gửi dữ liệu xuống cho một trạm (STA). Bảng 2.4. cho biết ý nghĩa của từng trường địa chỉ

*Bảng 2.4. Ý nghĩa của trường địa chỉ khi ToDS = 0, FromDS = 1*

Trường	Vai trò	Diễn giải
Address 1	RA = DA	Địa chỉ MAC của trạm nhận
Address 2	TA = BSSID	Địa chỉ MAC của AP, là bên phát sóng radio trực tiếp
Address 3	SA	Địa chỉ MAC của nguồn gốc (thiết bị đã gửi gói tin này cho AP, thường là router).
Address 4	Không dùng	

#### 4. Trường hợp 4: To DS = 1, From DS = 1

Khung được sử dụng trong chế độ cầu nối không dây (Wireless Distribution System - WDS), khi một AP chuyển tiếp khung dữ liệu cho một AP khác qua kết nối không dây. Đây là trường hợp duy nhất sử dụng cả 4 địa chỉ. Bảng 2.5. cho biết ý nghĩa của từng trường địa chỉ

*Bảng 2.5. Ý nghĩa của trường địa chỉ khi ToDS = 1, FromDS = 1*

Trường	Vai trò	Diễn giải
Address 1	RA	Địa chỉ MAC của AP nhận
Address 2	TA	Địa chỉ MAC của AP gửi (AP trung gian)
Address 3	DA	Địa chỉ MAC của điểm đích cuối cùng trong mạng
Address 4	SA	Địa chỉ MAC của thiết bị đã tạo ra gói tin

## 2.2.5 CSI

### 2.2.5.1 RSSI là gì?

Trong truyền thông không dây, kênh (channel) đại diện cho môi trường vật lý mà tín hiệu truyền qua từ máy phát đến máy thu. Các thuộc tính của kênh này quyết định chất lượng và độ tin cậy của đường truyền. Trong nhiều năm, chỉ số chính được sử dụng để mô tả đường truyền này là cường độ tín hiệu thu (Received Signal Strength Indicator - RSSI).

RSSI là một giá trị vô hướng, duy nhất, có đơn vị là dBm, đại diện cho tổng công suất của tín hiệu mà thiết bị thu được. Mặc dù dễ dàng thu được từ hầu hết mọi thiết bị không dây, RSSI về cơ bản là một chỉ số thô. Nó tổng hợp công suất của tín hiệu trên toàn bộ băng thông của kênh, không nắm bắt được các hiệu ứng phức tạp của hiện tượng truyền lan đa đường (multipath propagation) — tức là khi tín hiệu đến máy thu qua nhiều đường khác nhau do phản xạ, tán xạ và nhiễu xạ. Điều này làm cho RSSI rất nhạy cảm với các biến động theo thời gian và nhiễu, hạn chế công dụng của nó trong

các tác vụ cơ bản như ước tính cường độ tín hiệu và phát hiện sự hiện diện ở mức độ thô.

### 2.2.5.2 CSI là gì?

Thông tin trạng thái kênh (Channel State Information - CSI) không giống như giá trị đơn lẻ của RSSI, nó cung cấp một mô tả đa chiều, chi tiết về các thuộc tính của kênh.

Trong các hệ thống Wi-Fi hiện đại sử dụng kỹ thuật Orthogonal Frequency Division Multiplexing (OFDM) — chẳng hạn như các hệ thống tuân thủ tiêu chuẩn IEEE 802.11a/g/n/ac/ax—toàn bộ kênh truyền được chia thành nhiều sóng mang con (subcarrier) trực giao. CSI cung cấp thông tin về biên độ (amplitude) và pha (phase) cụ thể cho từng sóng mang con này. Điều này làm cho CSI trở thành một chỉ số ổn định và giàu thông tin hơn nhiều so với RSSI và rất có giá trị cho các ứng dụng cảm biến Wi-Fi (Wi-Fi sensing).

Để đo lường kênh truyền, thiết bị gửi sẽ gửi một tín hiệu đã được định nghĩa trước, được gọi là LTF (Long Training Field), ở phần đầu của một khung bản tin Wi-Fi. Vì thiết bị thu đã biết tín hiệu này đáng lẽ phải trông như thế nào, nó có thể đo lường sự khác biệt một cách chính xác.

Cụ thể, tín hiệu nhận được là sự kết hợp của tín hiệu gốc đã bị biến dạng bởi môi trường cộng với một số nhiễu điện tử ngẫu nhiên. Kênh truyền có thể được mô hình hóa như một hệ thống tuyến tính, biến đổi theo thời gian. Mỗi quan hệ giữa tín hiệu truyền và tín hiệu nhận có thể được mô tả bằng phương trình (2.1)

$$y = Hx + n \quad (2.1)$$

Với  $y$  là vectơ của các tín hiệu nhận được trên các anten thu,  $x$  là vectơ của các tín hiệu được truyền từ các anten phát,  $n$  là vectơ nhiễu và  $H$  là ma trận thông tin trạng thái kênh. Trong hệ thống MIMO-OFDM có  $N_t$  anten phát,  $N_r$  anten thu và  $K$  sóng mang con, kênh được mô tả bằng ma trận 3 chiều  $N_r * N_t * K$  gồm các số phức. Mỗi phần tử  $H_{ij,k}$  trong ma trận này biểu diễn độ lợi kênh (channel gain) phức ở dạng biên độ và pha giữa ăng ten phát thứ  $j$  và ăng ten thu thứ  $i$  trên sóng mang phụ thứ  $k$

Phương trình (2.2) biểu diễn một giá trị CSI trong ma trận (Ma, Zhou, & Wang, 2019)

$$H_{ij,k} = |H_{ij,k}|e^{j\angle H_{ij,k}} \quad (2.2)$$

### 2.2.5.3 Trích xuất CSI

Các card Wi-Fi thương mại thông thường (Commercial Off-The-Shelf) chủ yếu được thiết kế cho mục đích kết nối mạng và không tiết lộ thông tin ở tầng vật lý cấp

thấp. Vì vậy, một hệ sinh thái đặc thù đã được hình thành, bao gồm phần cứng chuyên dụng, firmware tùy chỉnh và các công cụ phần mềm, chủ yếu do cộng đồng nghiên cứu học thuật phát triển, nhằm khai thác và thu thập CSI (Channel State Information) từ các card Wi-Fi phổ biến.

Trong quá trình phát triển, nhiều công cụ đã được tạo ra để thu thập CSI. Một trong những công cụ đầu tiên là Linux 802.11n CSI Tool, hỗ trợ card Intel Wi-Fi Link 5300, đã đánh dấu bước khởi đầu quan trọng trong lĩnh vực này khi cho phép thu thập CSI từ mạng 802.11n. Cùng với sự phát triển của các chuẩn Wi-Fi hiện đại hơn, các công cụ mới như Atheros CSI và Nexmon CSI ra đời, khai thác thêm dữ liệu từ các sóng mang con và băng thông mở rộng, phục vụ các ứng dụng cảm biến không dây độ phân giải cao. Atheros CSI hỗ trợ các chipset Qualcomm Atheros, còn Nexmon CSI hướng tới các chip Broadcom hiện đại như trên Raspberry Pi và các thiết bị di động, tương thích với chuẩn 802.11ac và hỗ trợ tối đa 256 sóng mang con.

Gần đây, lĩnh vực nghiên cứu này phát triển nhanh chóng nhờ các nền tảng phần mềm trung gian như PicoScenes, hỗ trợ nhiều dòng card mạng và các chuẩn Wi-Fi mới nhất như 802.11ac/ax, bao gồm cả chipset Intel hiện đại. Ngoài ra, công cụ ESP32 CSI giúp thu thập CSI từ vi điều khiển ESP32 giá rẻ, góp phần mở rộng khả năng ứng dụng trong các hệ thống IoT tiết kiệm chi phí.

Đáng chú ý, card mạng Intel AX200/AX210 hiện rất phổ biến trên các laptop nhờ giá thành rẻ và khả năng tương thích cao. Dù PicoScenes có thể thu thập CSI từ các card này, nhưng đây là công cụ mã nguồn đóng và có tính phí. Vì vậy, trong đồ án này, nhóm sử dụng một driver được chỉnh sửa từ dự án mã nguồn mở FeitCSI – cho phép thu thập CSI từ card Intel AX200/AX210, hỗ trợ nhiều chuẩn Wi-Fi như 802.11a/g/n/ac/ax.

## **2.3 Netlink & nl80211**

Với hệ thống dò quét Wi-Fi được triển khai trong đồ án, Netlink là công cụ nền tảng và không thể thiếu. Nó cho phép ứng dụng (được viết bằng ngôn ngữ C++) tương tác sâu với card Wi-Fi ở mức độ thấp mà không cần gọi các công cụ dòng lệnh bên ngoài (như iw, ifconfig). Công cụ này hỗ trợ việc lấy thông tin của các card Wi-Fi, gửi yêu cầu đến driver của chúng để chuyển qua các kênh Wi-Fi khác nhau.

### **2.3.1 Netlink là gì?**

Netlink là một cơ chế giao tiếp liên tiến trình (Inter-Process Communication) được sử dụng trong nhân Linux để truyền thông tin giữa các thành phần trong không gian nhân (kernel space) và các tiến trình trong không gian người dùng (user space). Về bản chất, nó là một loại socket đặc biệt (AF\_NETLINK) được thiết kế riêng cho mục đích quản lý và cấu hình mạng, thay thế cho cơ chế ioctl truyền thống vốn phức tạp và kém linh hoạt hơn.

### 2.3.2 nl80211

Generic Netlink (genl) là một phiên bản mở rộng và linh hoạt hơn của netlink, cho phép định nghĩa các "gia đình" (families) lệnh con. nl80211 chính là một gia đình bên trong Generic Netlink, được thiết kế chuyên để cấu hình và tương tác với các thiết bị mạng không dây chuẩn IEEE 802.11. Nó cung cấp các API cho các tiến trình ở không gian người dùng để điều khiển các card Wi-Fi được gắn trên thiết bị.

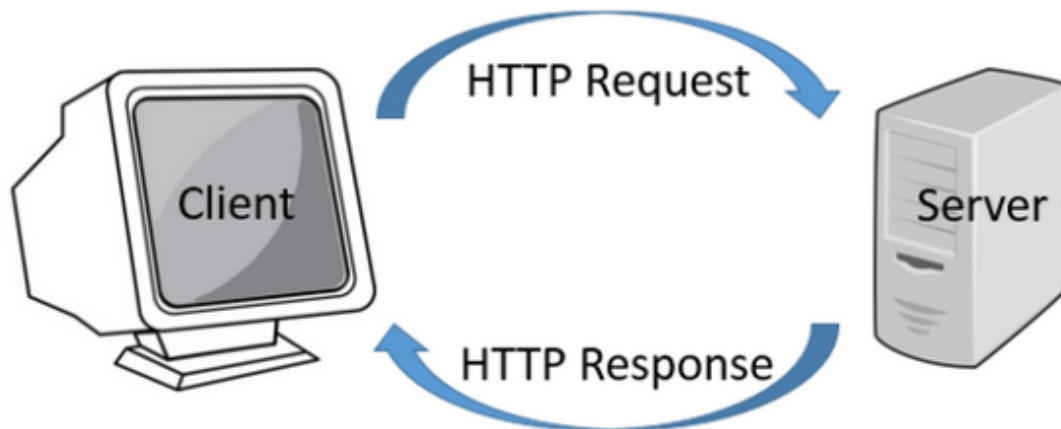
## 2.4 Các giao thức khác được sử dụng

### 2.4.1 Giao thức HTTP

HTTP (Hypertext Transfer Protocol) là một giao thức (quy tắc truyền tin) để trao đổi thông tin giữa máy chủ Web và trình duyệt Web. Khi chúng ta thu thập thông tin trên trang chủ của Web hoặc đọc các blog, HTTP được sử dụng để giao tiếp giữa máy chủ và máy khách (người dùng) [1].

#### 2.4.1.1 Mô hình client-server

HTTP hoạt động dựa trên mô hình client-server. Máy khách (client), thường là trình duyệt web hoặc ứng dụng, gửi một yêu cầu HTTP (HTTP request) tới máy chủ (server). Máy chủ nhận yêu cầu, xử lý và trả về một phản hồi HTTP (HTTP response) cho máy khách. Quá trình này tạo nên chu trình giao tiếp kiểu yêu cầu - phản hồi giữa hai bên



Hình 2.5. Quá trình giao tiếp giữa Client và Server bằng HTTP

#### 2.4.1.2 Thiết lập kết nối

Trước khi gửi yêu cầu HTTP, máy khách cần thiết lập kết nối TCP với máy chủ. Quá trình này thường bắt đầu bằng việc máy khách sử dụng Domain Name System để phân giải tên miền thành địa chỉ IP, sau đó thực hiện “bắt tay ba bước” (three-way handshake) để thiết lập kết nối TCP. Kết nối TCP đảm bảo dữ liệu được truyền tải tin cậy giữa client và server

#### 2.4.1.3 Gửi yêu cầu (HTTP Request)

Một HTTP request gồm các thành phần chính:



- Request line: Chứa phương thức HTTP (GET, POST, PUT, DELETE...), đường dẫn tài nguyên (URL) và phiên bản HTTP
- Headers: Các trường thông tin bổ sung như kiểu dữ liệu, xác thực, điều khiển cache, v.v..
- Body: Chứa dữ liệu gửi lên server (thường dùng với POST, PUT), có thể là biểu mẫu, file hoặc dữ liệu khác.

Các phương thức HTTP phổ biến gồm:

- GET: Yêu cầu lấy dữ liệu từ server.
- POST: Gửi dữ liệu lên server.
- PUT: Cập nhật dữ liệu.
- DELETE: Xóa tài nguyên

#### **2.4.1.4 Xử lý yêu cầu và phản hồi (HTTP Response)**

Server nhận yêu cầu, phân tích và thực hiện các hành động như trả về tài nguyên, thực thi chương trình, hoặc trả lỗi nếu không thể đáp ứng. Phản hồi HTTP gồm:

- Status line: Phiên bản HTTP, mã trạng thái (ví dụ 200 OK, 404 Not Found)
- Headers: Thông tin bổ sung về phản hồi, như loại nội dung, độ dài, v.v..
- Body: Dữ liệu nội dung trả về (HTML, JSON, hình ảnh, v.v.)

#### **2.4.1.5 Đóng hoặc tái sử dụng kết nối**

Ở HTTP/1.0, mỗi yêu cầu - phản hồi sử dụng một kết nối TCP riêng biệt, sau khi hoàn thành sẽ đóng kết nối. HTTP/1.1 giới thiệu cơ chế persistent connections (keep-alive), cho phép tái sử dụng kết nối TCP cho nhiều yêu cầu - phản hồi, giúp giảm độ trễ và tăng hiệu suất. HTTP/2 cải thiện hiệu suất bằng cách sử dụng dữ liệu nhị phân, nén header, hỗ trợ đa luồng (multiplexing), và cơ chế server push giúp giảm độ trễ tải trang.

#### **2.4.1.6 Đặc điểm của HTTP**

HTTP là giao thức stateless (không lưu trạng thái): server không lưu thông tin trạng thái giữa các yêu cầu, mỗi yêu cầu được xử lý độc lập. HTTP cũng là giao thức connectionless (không duy trì kết nối lâu dài): mỗi cặp yêu cầu - phản hồi hoạt động độc lập, kết nối TCP có thể được đóng sau khi hoàn thành

#### 2.4.1.7 Tóm tắt cơ chế hoạt động của HTTP

- Client gửi yêu cầu HTTP tới server qua kết nối TCP.
- Server xử lý yêu cầu, trả về phản hồi HTTP.
- Kết nối TCP có thể được tái sử dụng hoặc đóng tùy phiên bản HTTP.
- HTTP là giao thức không lưu trạng thái, mỗi yêu cầu độc lập.
- Các phương thức HTTP xác định hành động client muốn server thực hiện (GET, POST, PUT, DELETE,...).
- HTTP hoạt động dựa trên mô hình yêu cầu - phản hồi giữa client và server

Như vậy, cơ chế của HTTP là một chu trình gồm thiết lập kết nối TCP, gửi yêu cầu HTTP, xử lý và trả về phản hồi, rồi đóng hoặc tái sử dụng kết nối, với đặc điểm chính là giao thức không lưu trạng thái và hoạt động theo mô hình client-server dựa trên các phương thức HTTP định nghĩa sẵn.

#### 2.4.2 Giao thức WebSocket

WebSocket là một giao thức truyền thông máy tính (computer communication protocol), cung cấp các kênh liên lạc dạng *full-duplex* (song công) qua một kết nối TCP. Sử dụng WebSocket, bạn có thể tạo ra những ứng dụng *realtime* thật sự như chat, chỉnh sửa tài liệu online (ví dụ Google Docs), giao dịch hoặc game online nhiều người chơi [2].

##### 2.4.2.1 Lý do WebSocket xuất hiện

Hệ thống liên lạc song công (duplex) là một hệ thống điểm-điểm bao gồm hai hoặc nhiều bên được kết nối có thể giao tiếp với nhau theo cả hai hướng. Có hai loại:

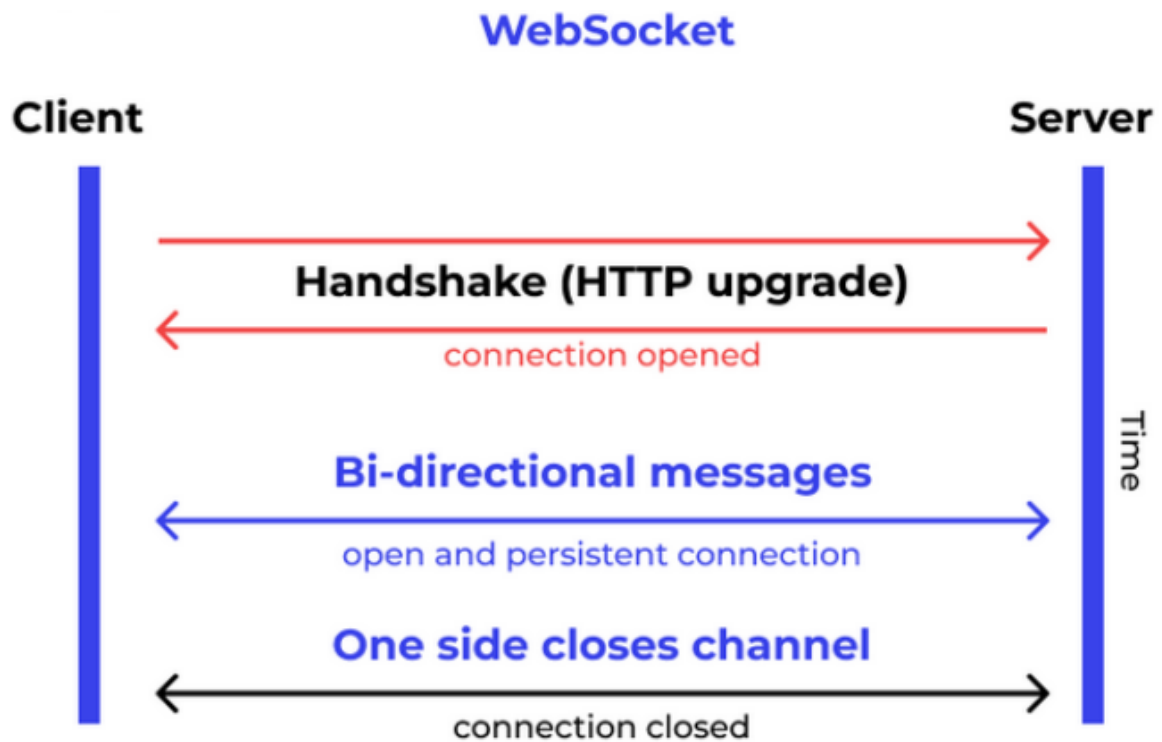
- Song công hoàn toàn (Full-duplex): Giao tiếp đồng thời hai chiều.
- Bán song công (Half-duplex): Giao tiếp hai chiều nhưng không đồng thời.

Trong mô hình client-server truyền thống sử dụng HTTP/1.0 hoặc HTTP/1.1, client gửi yêu cầu và server phản hồi. Dù HTTP/1.1 đã cải thiện bằng kết nối tái sử dụng (persistent connection), bản chất của HTTP vẫn là half-duplex: dữ liệu chỉ được truyền theo một chiều tại một thời điểm, gây ra lãng phí và độ trễ.

##### Sử dụng WebSocket để giảm overhead

WebSocket thiết lập một kết nối TCP duy nhất và sau đó thực hiện trao đổi dữ liệu song công toàn phần mà không phải thiết lập lại kết nối liên tục như HTTP. Điều này giảm đáng kể overhead do việc thiết lập và đóng kết nối liên tục trong HTTP.

#### 2.4.2.2 Cách hoạt động của WebSocket



Hình 2.6. Quá trình giao tiếp giữa Client và Server bằng WebSocket

##### 1. Bắt tay (Handshake):

- Kết nối bắt đầu bằng một yêu cầu HTTP đặc biệt với header Upgrade để chuyển sang giao thức WebSocket.
- Một số header quan trọng: Sec-WebSocket-Key, Sec-WebSocket-Accept, Sec-WebSocket-Version.
- Sau khi server chấp nhận, kết nối được nâng cấp và chuyển sang WebSocket.

##### 2. Truyền dữ liệu:

- Sau handshake, client và server có thể gửi/nhận dữ liệu bất kỳ lúc nào.
- Dữ liệu được đóng gói thành các frame, có thể là text hoặc binary.
- Hỗ trợ các extension (nén dữ liệu, bảo mật) và subprotocol (giao thức ứng dụng chạy trên WebSocket).

##### 3. Đóng kết nối:

- Một trong hai phía có thể chủ động đóng kết nối bằng một handshake đóng (closing handshake).

### 2.4.2.3 Một số điểm lưu ý khi triển khai WebSocket

- WebSocket cần có hỗ trợ từ cả client và server.
- Do hoạt động ở tầng TCP, nên cần xử lý thêm các vấn đề bảo mật như SSL/TLS (WebSocket Secure - wss://).
- Cần quản lý trạng thái kết nối để đảm bảo kết nối được duy trì hoặc khôi phục khi bị mất.

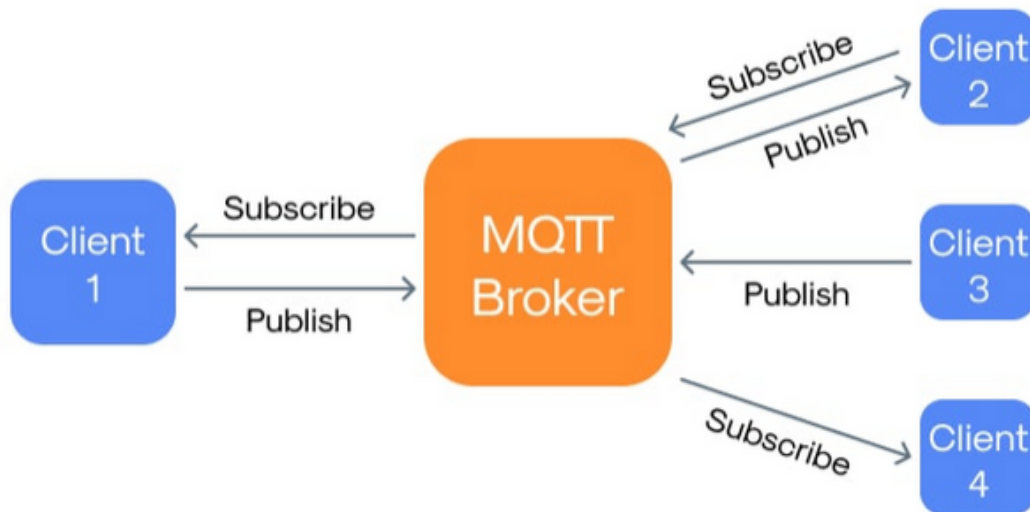
### 2.4.3 Giao thức MQTT

MQTT (Message Queuing Telemetry Transport) là một giao thức nhắn tin theo mô hình xuất bản/đăng ký (publish/subscribe), được thiết kế tối ưu cho các thiết bị IoT, đặc biệt trong môi trường băng thông thấp, độ trễ cao hoặc mạng không ổn định. MQTT nổi bật nhờ sự gọn nhẹ, tiêu tốn ít tài nguyên và khả năng truyền thông tin hiệu quả giữa hàng triệu thiết bị thông minh, cảm biến, vi điều khiển, v.v [3].

#### 2.4.3.1 Đặc điểm nổi bật của MQTT:

- Gọn nhẹ và hiệu quả: Sử dụng kích thước gói tin nhỏ, tiêu tốn ít băng thông và tài nguyên, phù hợp cho thiết bị IoT hạn chế về bộ nhớ, năng lượng [4].
- Mô hình publish/subscribe: Các thiết bị (client) xuất bản (publish) hoặc đăng ký (subscribe) vào các chủ đề (topic) thông qua một máy chủ trung gian gọi là broker. Broker chịu trách nhiệm chuyển tiếp thông điệp giữa các client mà không cần hai bên biết trực tiếp về nhau.
- Độ tin cậy cao: Hỗ trợ nhiều mức chất lượng dịch vụ (QoS) để đảm bảo thông điệp được gửi và nhận đúng, kể cả khi mạng không ổn định.
- Bảo mật: Hỗ trợ mã hóa dữ liệu qua SSL/TLS, xác thực người dùng và thiết bị bằng username/password hoặc chứng chỉ số.
- Khả năng mở rộng: Có thể triển khai cho hệ thống với hàng triệu thiết bị kết nối đồng thời.

# MQTT Protocol



Hình 2.7. Quá trình giao tiếp giữa Client và Broker bằng MQTT

## 2.4.3.2 Cách thức hoạt động của MQTT:

1. Kết nối: Client thiết lập kết nối TCP tới broker (cổng mặc định 1883 cho không mã hóa, 8883 cho mã hóa SSL/TLS).
2. Xác thực: Client có thể gửi thông tin xác thực (username/password hoặc chứng chỉ) để broker kiểm tra quyền truy cập.
3. Xuất bản (Publish): Client gửi thông điệp lên một chủ đề (topic) cụ thể. Broker sẽ chuyển tiếp thông điệp này tới tất cả các client đã đăng ký (subscribe) chủ đề đó.
4. Đăng ký (Subscribe): Client đăng ký nhận thông điệp từ một hoặc nhiều chủ đề. Khi có thông điệp mới trên chủ đề đó, broker sẽ chuyển tiếp tới client.
5. Ngắt kết nối: Client có thể chủ động ngắt kết nối hoặc broker sẽ ngắt khi client không còn hoạt động.

## 2.4.3.3 Ứng dụng thực tiễn:

- Giám sát và điều khiển thiết bị IoT, nhà thông minh
- Truyền dữ liệu cảm biến trong công nghiệp, nông nghiệp
- Giao tiếp giữa các thiết bị nhúng, thiết bị di động
- Ứng dụng nhắn tin thời gian thực (ví dụ: Facebook Messenger cũng sử dụng MQTT)

#### 2.4.3.4 Tóm tắt ưu điểm của MQTT:

- Tiết kiệm tài nguyên, phù hợp cho thiết bị nhỏ, tiết kiệm năng lượng
- Đảm bảo truyền thông tin tin cậy, kể cả khi mạng không ổn định
- Dễ dàng mở rộng quy mô hệ thống IoT
- Hỗ trợ bảo mật mạnh mẽ với SSL/TLS và xác thực nhiều lớp

MQTT là giao thức lý tưởng cho các hệ thống IoT nhờ tính gọn nhẹ, hiệu quả, dễ mở rộng và độ tin cậy cao, giúp đơn giản hóa việc truyền thông giữa các thiết bị thông minh trong nhiều lĩnh vực khác nhau.

### 2.5 Kiến thức Web cơ bản

#### 2.5.1 SPA (Single Page Application)

Single Page Application (SPA) là loại ứng dụng web hỗ trợ tương tác và không yêu cầu người dùng tải lại trang mỗi khi truy cập. Thay vào đó, nền tảng sẽ tải ứng dụng thông qua một quá trình duy nhất, khi mà người dùng mới truy cập lần đầu. SPA thường sử dụng các kỹ thuật như AJAX để tải dữ liệu và cập nhật giao diện người dùng mà không cần tải lại toàn bộ trang web [5].

- Tại sao Single Page Application được phát triển?

Single Page Application (SPA) xuất phát từ nhu cầu tăng cường trải nghiệm của người dùng và cải thiện hiệu suất của web ứng dụng. Hệ thống truyền tải web ứng dụng thường yêu cầu tải lại toàn bộ trang khi người dùng tương tác. Điều này có thể dẫn đến thời gian tải xuống lâu, trải nghiệm người dùng không mượt mà và vấn đề thay đổi trạng thái ngẫu nhiên của ứng dụng.

- Sự khác biệt giữa web SPA và web truyền thống?

– Tải trang ban đầu:

- \* Web SPA: Khi người dùng truy cập, toàn bộ ứng dụng chỉ được tải duy nhất một lần. Sau đó, khi người dùng tương tác, nội dung mới có thể được tải mà không cần phải tải lại toàn bộ trang.
- \* Trang web truyền thống: Mỗi khi người dùng tương tác (ví dụ như nhấn nút hoặc thực hiện hành động), trang web cần tải lại toàn bộ trang, đôi khi dẫn đến mất điểm trải nghiệm người dùng và làm tăng thời gian tải trang.

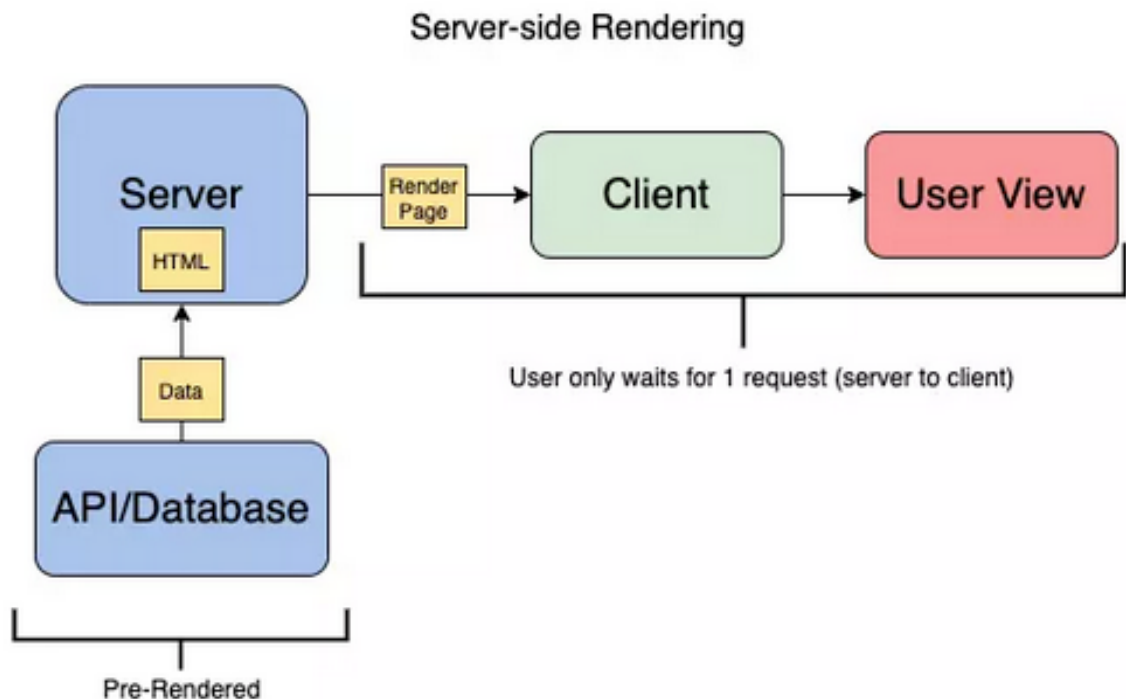
– Tương tác dữ liệu:

- \* Web SPA: Sử dụng kỹ thuật AJAX để tải và cập nhật dữ liệu mà không cần tải lại toàn bộ trang web khi có thay đổi.

- \* Trang web truyền thống: Yêu cầu tải lại trang để cập nhật dữ liệu, dẫn đến trải nghiệm người dùng không mượt mà.
- Quản lý trạng thái ứng dụng:
  - \* Web SPA: Sử dụng các framework và thư viện như Angular, React, Vue.js để quản lý trạng thái ứng dụng một cách linh hoạt và hiệu quả.
  - \* Trang web truyền thống: Thường không có cơ chế quản lý trạng thái tốt, dẫn đến vấn đề quản lý ứng dụng phức tạp hơn.
- Hiệu suất và trải nghiệm người dùng:
  - \* Web SPA: Mang lại trải nghiệm người dùng mượt mà hơn do không cần tải lại toàn bộ trang, đồng thời có khả năng tương tác nhanh hơn và phản ứng linh hoạt hơn.
  - \* Trang web truyền thống: Thường có thời gian tải trang dài hơn và trải nghiệm người dùng không mượt mà do phải tải lại toàn bộ trang.

### 2.5.2 SSR (Server Side Rendering) và CSR (Client Side Rendering)

- SSR (Server Side Rendering) Từ những năm 2000, SSR đã được sử dụng rất phổ biến, gọi nó là SSR vì hầu hết các logic phức tạp trên trang web sẽ được xử lý ở phía server [6].

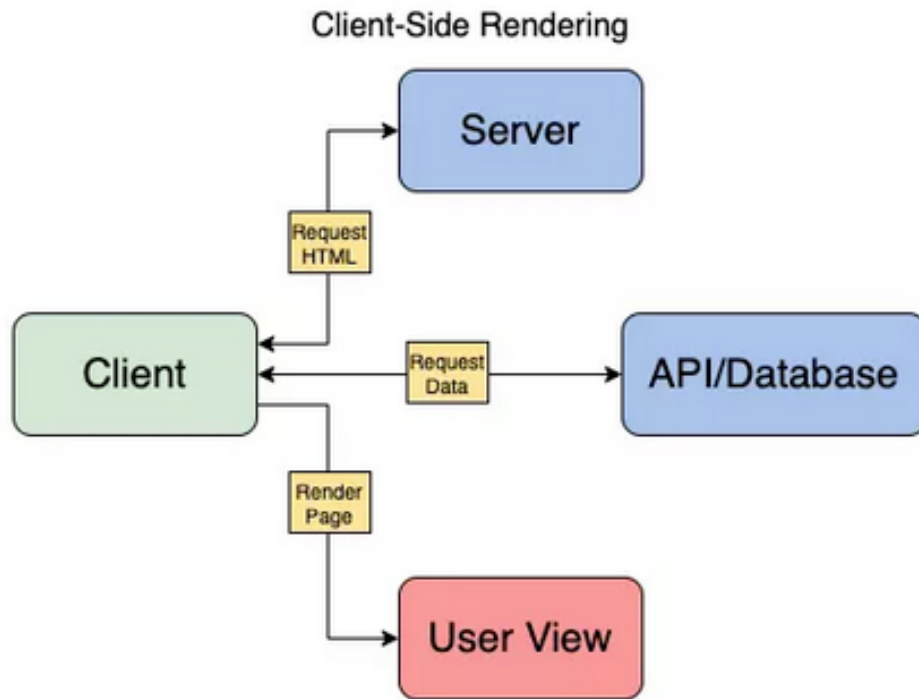


Hình 2.8. Server Side Rendering

- Cơ chế của SSR như sau:
  - \* Khi user truy cập vào một trang web, trình duyệt sẽ gửi request tới server

- \* Server sẽ nhận request, truy cập vào dữ liệu trong database, render ra HTML
  - \* Trả HTML kèm với CSS, JS về cho browser
  - \* Browser nhận được HTML thì tiến hành tải xuống và render ra UI nhưng lúc này chưa có JS
  - \* Browser tải xuống JS và thực thi các câu lệnh JS
  - \* Website load xong và có thể tương tác bình thường
- Ưu điểm của SSR:
- \* Load lần đầu nhanh vì toàn bộ dữ liệu đã được xử lý ở phía server, client chỉ cần nhận về và hiển thị ra cho user
  - \* Tốt cho SEO vì dữ liệu được render dưới dạng HTML (bạn ấn chuột phải chọn View Page Source một trang web bất kì là sẽ thấy điều này), giúp cho bot của Google khi quét sẽ thấy được toàn bộ dữ liệu
  - \* Lập trình viên chỉ cần code trên 1 project là đã có thể tạo ra trang web hoàn chỉnh có cả Frontend lẫn Backend, không cần tách ra làm 2 project.
- Nhược điểm của SSR:
- \* Server sẽ phải xử lý nhiều dữ liệu dẫn đến quá tải
  - \* Khi user chuyển trang thì cả trang sẽ phải load lại để lấy dữ liệu từ server, dẫn đến trải nghiệm không tốt
- CSR (Client Side Rendering) Sở dĩ nó được gọi là CSR vì việc render HTML sẽ được thực thi ở phía client. Hay chúng ta còn gọi là Single Page App (SPA) [7].





*Hình 2.9. Client Side Rendering*

– Cơ chế của CSR như sau:

- \* Khi user truy cập vào một trang web, trình duyệt sẽ gửi request tới server.
- \* Server trả về một file HTML tối thiểu (chủ yếu là khung chứa và link tới các file JS, CSS).
- \* Browser tải xuống và thực thi các file CSS và JS.
- \* JS sẽ gửi các request đến API hoặc database để lấy dữ liệu cần thiết.
- \* Sau khi nhận dữ liệu, JS sẽ render nội dung giao diện (UI) trực tiếp trên trình duyệt.
- \* Website bắt đầu hiển thị và người dùng có thể tương tác.

– Các ưu điểm của CSR:

- \* Chuyển việc xử lý dữ liệu sang cho client giúp server nhẹ việc hơn
- \* Trang chỉ cần load một lần duy nhất, khi user muốn lấy dữ liệu mới từ server chỉ cần gọi đến server thông qua AJAX
- \* Trang web không cần load lại nhiều khi user chuyển trang, đem đến trải nghiệm tốt hơn cho người dùng

– Tuy nhiên, CSR vẫn còn một số nhược điểm như dưới đây:

- \* Load trang lần đầu sẽ chậm hơn SSR nếu không tối ưu. Do browser phải tải toàn bộ HTML và JS ở lần đầu load trang, chạy JS và gọi API để lấy dữ liệu từ server rồi mới render lên UI cho user.

- \* Lập trình viên phải chia ra làm 2 project: Backend để viết API và Frontend để hiển thị
- \* Website không thể chạy nếu tắt JavaScript ở trình duyệt
- \* Nếu user sử dụng thiết bị có cấu hình yếu thì sẽ bị load chậm
- \* SEO không tốt bằng SSR do dữ liệu được render bởi JS là dữ liệu động

## 2.6 Các công nghệ liên quan

### 2.6.1 *ThingsBoard Platform*

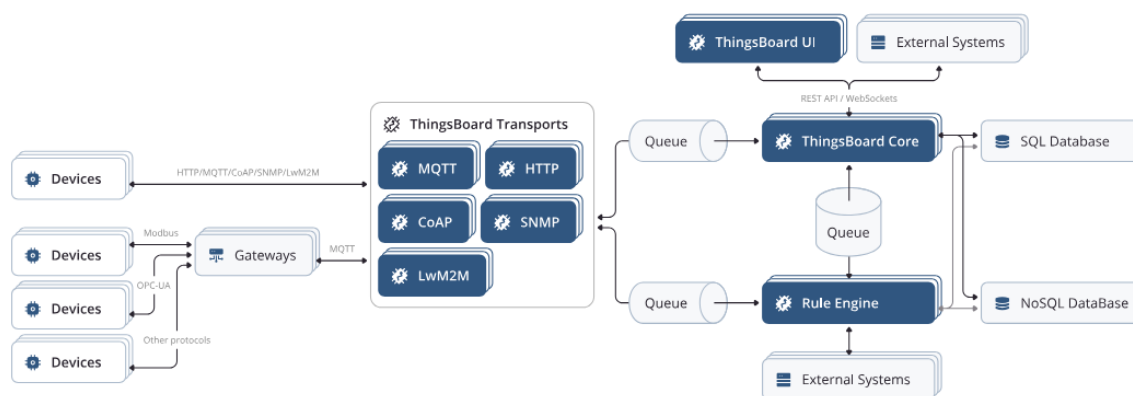
#### 2.6.1.1 Tổng quan về ThingsBoard

Thingsboard là một nền tảng mã nguồn mở để quản lý thiết bị IoT (Internet of Things), thu thập dữ liệu từ thiết bị, xử lý và trực quan hóa dữ liệu thời gian thực. ThingsBoard hỗ trợ các giao thức phổ biến như MQTT và HTTP, đồng thời cung cấp cơ chế quản lý thiết bị, người dùng và các quy tắc xử lý dữ liệu một cách linh hoạt và mở rộng dễ dàng. Đây là nền tảng phù hợp để xây dựng các hệ thống IoT công nghiệp, giám sát môi trường, nhà thông minh, v.v [8]. ThingsBoard được thiết kế với các đặc điểm nổi bật như sau:

- Khả năng mở rộng: Nền tảng có thể mở rộng theo chiều ngang, được xây dựng dựa trên các công nghệ mã nguồn mở hàng đầu.
- Chống lỗi (Fault-tolerant): Không có điểm lỗi đơn lẻ, mọi nút trong cụm (cluster) đều giống nhau và có thể thay thế cho nhau.
- Mạnh mẽ và hiệu quả: Một máy chủ đơn lẻ có thể xử lý hàng chục đến hàng trăm nghìn thiết bị tùy theo trường hợp sử dụng. Cụm ThingsBoard có thể xử lý hàng triệu thiết bị.
- Độ bền cao (Durability): Không làm mất dữ liệu. ThingsBoard hỗ trợ nhiều cách triển khai hàng đợi (queue) khác nhau để đảm bảo độ bền cực cao cho các thông điệp.
- Tùy biến: Dễ dàng mở rộng chức năng mới thông qua các tiện ích (widget) và các nút của Rule Engine có thể tùy biến.

#### 2.6.1.2 Kiến trúc hệ thống

Sơ đồ dưới đây thể hiện các thành phần chính của hệ thống và các giao diện mà chúng cung cấp:



Hình 2.10. Kiến trúc của ThingsBoard

#### a. ThingsBoard Transports

ThingsBoard cung cấp các API dựa trên MQTT, HTTP, CoAP... cho các ứng dụng/firmware thiết bị. Mỗi giao thức được xử lý bởi một thành phần máy chủ riêng biệt trong lớp “Transport Layer”. Transport MQTT cũng cung cấp API cho Gateway – dùng để đại diện cho nhiều thiết bị hoặc cảm biến được kết nối.

Khi Transport nhận được thông điệp từ thiết bị, nó sẽ phân tích cú pháp và đẩy thông điệp vào hàng đợi tin nhắn bền vững (durable Message Queue). Thiết bị chỉ nhận được xác nhận sau khi thông điệp đó đã được hàng đợi xử lý thành công.

#### b. ThingsBoard Core

ThingsBoard Core chịu trách nhiệm xử lý các REST API và đăng ký WebSocket. Nó cũng lưu trữ trạng thái phiên hoạt động của thiết bị và giám sát kết nối của thiết bị. Core sử dụng Actor System để triển khai các actor cho các thực thể chính như tenants và devices. Các node trong nền tảng có thể tham gia vào một cụm (cluster), mỗi node chịu trách nhiệm với một phân vùng của thông điệp đến.

#### c. ThingsBoard Rule Engine

Rule Engine là thành phần trung tâm của hệ thống, xử lý các thông điệp đầu vào. Nó sử dụng Actor System để triển khai các actor cho rule chains và rule nodes. Rule Engine có thể mở rộng theo cụm, trong đó mỗi node phụ trách một phần dữ liệu đầu vào.

Rule Engine đăng ký luồng dữ liệu từ hàng đợi và chỉ xác nhận sau khi đã xử lý xong. Có nhiều chiến lược xử lý và xác nhận được hỗ trợ, có thể tùy chỉnh theo nhu cầu sử dụng.

Rule Engine có hai chế độ hoạt động: *shared* và *isolated*. Trong chế độ *shared*, nó xử lý dữ liệu từ nhiều tenants. Trong chế độ *isolated*, nó chỉ xử lý dữ liệu của một số tenant nhất định.

#### d. ThingsBoard Web UI

ThingsBoard cung cấp giao diện người dùng web nhẹ được xây dựng bằng framework Express.js để phục vụ nội dung tĩnh. Thành phần này không lưu trạng thái và không yêu cầu cấu hình phức tạp. Sau khi tải giao diện, ứng dụng sẽ sử dụng REST API và WebSocket API từ ThingsBoard Core để hoạt động.

#### e. ThingsBoard Message Queues

ThingsBoard hỗ trợ hai loại hàng đợi tin nhắn (message queues): Kafka và In-Memory.

- Kafka là một hệ thống hàng đợi tin nhắn phân tán, bền vững, được sử dụng rộng rãi, thiết kế để xử lý khối lượng lớn dữ liệu. Kafka rất phù hợp với môi trường sản xuất, nơi yêu cầu thông lượng cao, khả năng chịu lỗi và khả năng mở rộng.
- In-Memory queue là hàng đợi tin nhắn nhẹ, nhanh và đơn giản, được thiết kế cho mục đích thử nghiệm, phát triển hoặc các môi trường quy mô nhỏ. Hàng đợi này lưu trữ tin nhắn trong bộ nhớ thay vì trên đĩa, ưu tiên tốc độ hơn tính bền vững.

ThingsBoard sử dụng các topic sau:

- `tb_transport.api.requests`: gửi các API gọi chung để kiểm tra thông tin thiết bị từ Transport đến ThingsBoard Core.
- `tb_transport.api.responses`: nhận kết quả xác thực thiết bị từ ThingsBoard Core gửi về Transport.
- `tb_core`: đẩy tin nhắn từ Transport hoặc Rule Engine đến ThingsBoard Core, bao gồm các sự kiện vòng đời phiên, đăng ký thuộc tính và RPC, v.v.
- `tb_rule_engine`: đẩy tin nhắn từ Transport hoặc ThingsBoard Core đến Rule Engine, bao gồm dữ liệu telemetry đầu vào, trạng thái thiết bị, sự kiện vòng đời thực thể, v.v.

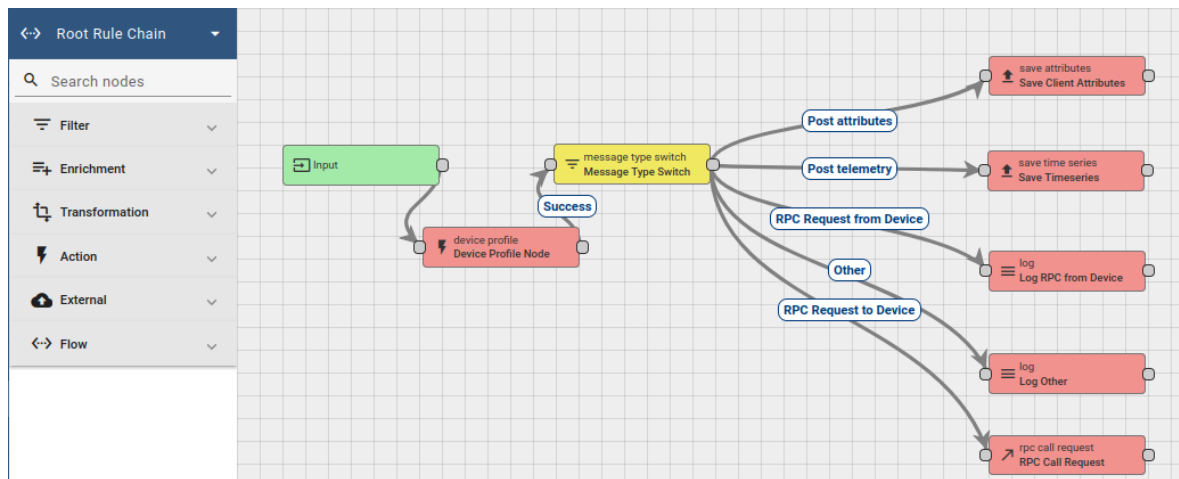
#### 2.6.1.3 ThingsBoard Rule Engine

Rule Engine là một framework để sử dụng để xây dựng các luồng công việc dựa trên sự kiện. Có ba thành phần chính:

- Message – đại diện cho mọi sự kiện đầu vào. Một Message có thể là:
  - Telemetry, Attribute hoặc RPC call gửi từ Device.
  - Sự kiện life-cycle của Entity: tạo, xóa, cập nhật, gán hoặc huỷ gán.
  - Sự kiện state thiết bị: kết nối, ngắt kết nối, hoạt động, không hoạt động.

Mỗi Message chứa các trường thông tin sau:

- *Message ID*: định danh duy nhất theo thời gian (UUID).
  - *Originator*: là Device, Asset, hay Entity tạo ra thông điệp.
  - *Type*: ví dụ “Post telemetry”, “Attributes Updated”, “Inactivity Event”.
  - *Payload*: phần nội dung chính của thông điệp dưới dạng JSON.
  - *Metadata*: danh sách các cặp key-value chứa dữ liệu bổ sung về thông điệp.
- Rule Node – đơn vị xử lý cơ bản trong Rule Engine. Mỗi Rule Node:
    - Xử lý một Message tại một thời điểm.
    - Các loại Rule Node gồm Filter Nodes, Enrichment Nodes, Transformation Nodes, Action Nodes, External Nodes, Flow Node giúp lọc, biến đổi, làm giàu thông điệp hoặc thực hiện hành động như:
      - \* Gửi REST API.
      - \* Gửi email.
      - \* Lưu dữ liệu vào cơ sở dữ liệu.
      - \* Kích hoạt hoặc xoá cảnh báo (alarm).
    - Các Rule Node được kết nối với nhau thông qua các quan hệ có nhãn như “Success”, “Failure”, “True”, “False” hoặc các nhãn tùy chỉnh.
  - Rule Chain – là tập hợp các Rule Node được kết nối theo luồng xử lý logic. Rule Chain cho phép:
    - Xây dựng luồng xử lý dữ liệu linh hoạt, dễ mở rộng.
    - Có một Rule Chain gốc (*Root Rule Chain*) nhận toàn bộ thông điệp đầu vào từ hệ thống.
    - Gọi tới các Rule Chain khác để chia nhỏ logic xử lý hoặc tái sử dụng.
    - Ví dụ ứng dụng:
      - \* Kiểm tra và hiệu chỉnh dữ liệu telemetry trước khi lưu.
      - \* Phát hiện và tạo cảnh báo nếu giá trị vượt ngưỡng.
      - \* Gửi thông báo đến người dùng tương ứng.
      - \* Giao tiếp với hệ thống ngoài như Kafka hoặc HTTP Endpoint.



Hình 2.11. Mẫu Root Rule Chain

### Các trường hợp sử dụng điển hình

ThingsBoard Rule Engine là một framework rất linh hoạt để xử lý các sự kiện phức tạp. Dưới đây là một số trường hợp sử dụng phổ biến có thể được cấu hình qua Rule Chains của ThingsBoard:

- Xác thực và chỉnh sửa dữ liệu telemetry hoặc thuộc tính đầu vào trước khi lưu vào cơ sở dữ liệu.
- Sao chép dữ liệu telemetry hoặc thuộc tính từ thiết bị sang các tài sản (Asset) liên quan để có thể tổng hợp dữ liệu. Ví dụ, dữ liệu từ nhiều thiết bị có thể được tổng hợp trong một Asset liên quan.
- Tạo, cập nhật hoặc xóa cảnh báo dựa trên các điều kiện đã định nghĩa.
- Kích hoạt các hành động dựa trên sự kiện vòng đời thiết bị. Ví dụ, tạo cảnh báo khi thiết bị Online hoặc Offline.
- Tải thêm dữ liệu cần thiết cho quá trình xử lý. Ví dụ, tải giá trị ngưỡng nhiệt độ của thiết bị được định nghĩa trong thuộc tính Customer hoặc Tenant.
- Gọi REST API tới các hệ thống bên ngoài.
- Gửi email khi xảy ra các sự kiện phức tạp và sử dụng thuộc tính của các thực thể khác trong mẫu email.
- Tính đến sở thích của người dùng trong quá trình xử lý sự kiện.
- Thực hiện các cuộc gọi RPC dựa trên điều kiện đã định nghĩa.
- Tích hợp với các pipeline bên ngoài như Kafka, Spark, các dịch vụ AWS, v.v.

#### 2.6.1.4 ThingsBoard API

Thingsboard hỗ trợ khá nhiều API phổ biến. ThingsBoard API bao gồm hai phần chính: Device API và server-side API.

- Device API – được chia theo các giao thức truyền thông được hỗ trợ:
  - MQTT API
  - MQTT Sparkplug API
  - CoAP API
  - HTTP API
  - LwM2M API
  - SNMP API
- Gateway MQTT API – cho phép kết nối các thiết bị hiện có vào nền tảng thông qua ThingsBoard Gateway hoặc thông qua gateway tùy biến riêng.
- Server-side API – được cung cấp dưới dạng REST API, bao gồm:
  - Administration REST API – cung cấp các API cốt lõi cho quản trị hệ thống.
  - Attributes Query API – API phía server từ dịch vụ Telemetry để truy vấn thuộc tính.
  - Timeseries Query API – API truy vấn dữ liệu chuỗi thời gian.
  - RPC API – API để gửi và nhận Remote Procedure Calls (RPC).
  - REST Client – thư viện khách sử dụng REST.
  - Python REST Client – thư viện REST client viết bằng Python.
  - Dart API Client – thư viện client dành cho ngôn ngữ Dart.

#### 2.6.2 PostgreSQL Database

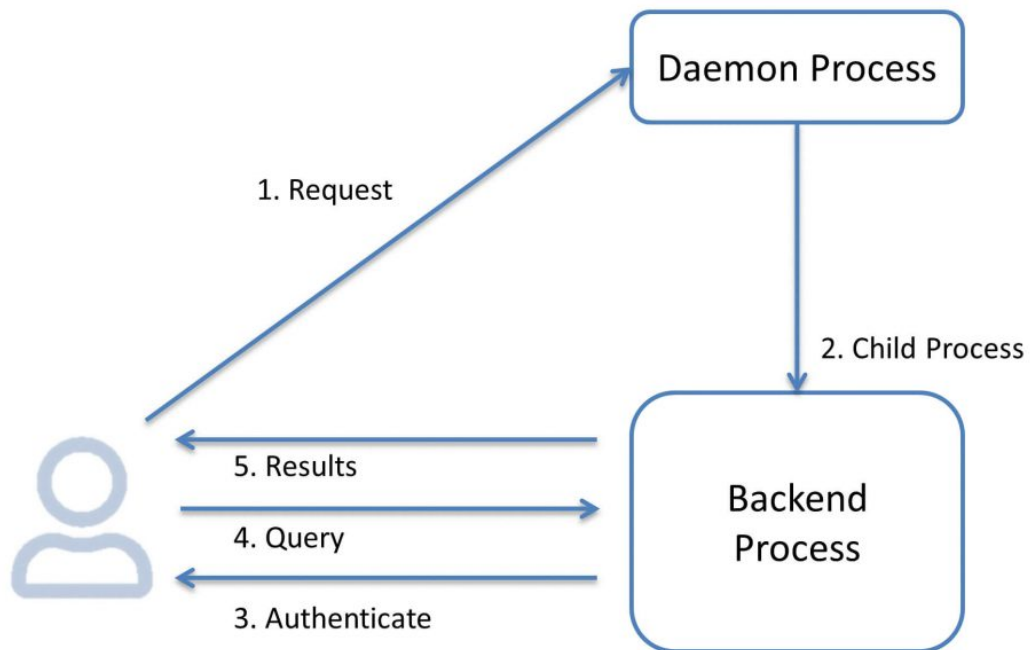
PostgreSQL là hệ quản trị cơ sở dữ liệu quan hệ mã nguồn mở mạnh mẽ, được lựa chọn làm nền tảng lưu trữ dữ liệu chính cho hệ thống. PostgreSQL cung cấp khả năng xử lý dữ liệu phức tạp với hiệu suất cao, hỗ trợ các tính năng nâng cao như JSON, full-text search, và khả năng mở rộng linh hoạt [9][10]. Trong dự án này, PostgreSQL được sử dụng để:

- Lưu trữ dữ liệu cảm biến thu thập từ các thiết bị IoT
- Quản lý thông tin người dùng và phân quyền truy cập
- Lưu trữ cấu hình hệ thống và metadata

- Hỗ trợ các truy vấn phức tạp cho báo cáo và phân tích dữ liệu

Việc sử dụng PostgreSQL đảm bảo tính toàn vẹn dữ liệu thông qua ACID compliance, khả năng backup và recovery tin cậy, cùng với hiệu suất tối ưu cho các ứng dụng có lượng truy cập cao.

### Client-Server Model



Hình 2.12. PostgreSQL Model

Postgres hoạt động theo kiến trúc khách – chủ (Client-Server), trong đó:

- Máy chủ Postgres(Server) luôn lắng nghe tại một cổng (port) xác định (mặc định là port 5432).
- Các chương trình khách(Client), chẳng hạn như textttpsql hoặc các dịch vụ ứng dụng, sẽ khởi tạo một *kết nối TCP* đến máy chủ Postgres để gửi yêu cầu truy xuất dữ liệu.

Khi một khách (Client) gửi yêu cầu:

- Máy chủ Postgres khởi một backend process riêng để xử lý khách này.
- Daemon process đóng vai trò tiếp nhận yêu cầu từ khách, khởi tạo một tiến trình con (backend) để xử lý.



- Tiến trình backend sau khi khởi tạo sẽ thực hiện xác thực khách, xử lý truy vấn, và trả về kết quả.

Như vậy, mỗi khi một khách nối đến, Postgres sẽ khởi một backend process riêng để xử lý, giúp tiện dụng trong quản lý tài nguyên, đồng thời tăng khả năng đồng thời khi nhiều khách cùng truy cập cơ sở dữ liệu.

### **2.6.3 FastAPI Framework**

FastAPI là một framework web hiện đại, hiệu suất cao được thiết kế để xây dựng API với Python 3.7+ dựa trên type hints chuẩn của Python. FastAPI trở thành một trong những framework Python phổ biến nhất cho việc phát triển API [11][12].

#### **2.6.3.1 Hiệu suất**

FastAPI được xây dựng trên hai nền tảng chính là Starlette và Pydantic. Starlette cung cấp khả năng xử lý bất đồng bộ thông qua ASGI (Asynchronous Server Gateway Interface), trong khi Pydantic đảm bảo validation và serialization dữ liệu mạnh mẽ. Framework này có hiệu suất ngang bằng với NodeJS và Go, trở thành một trong những framework Python nhanh nhất hiện có.

Khả năng xử lý bất đồng bộ của FastAPI cho phép xử lý hàng nghìn request đồng thời mà không tiêu tốn quá nhiều tài nguyên hệ thống. Điều này đạt được thông qua việc sử dụng async/await syntax để tạo ra các non-blocking I/O operations hiệu quả.

#### **2.6.3.2 Tính năng chính**

FastAPI cung cấp nhiều tính năng nổi bật:

- Automatic Documentation: Tự động tạo tài liệu API tương tác sử dụng OpenAPI (Swagger) và JSON Schema
- Type Validation: Sử dụng Python type hints để validation tự động và giảm thiểu lỗi runtime
- Fast Development: Tăng tốc độ phát triển tính năng
- Security Integration: Hỗ trợ đầy đủ các cơ chế authentication như OAuth2 và JWT
- WebSocket Support: Hỗ trợ real-time communication thông qua WebSocket connections

FastAPI tích hợp liền mạch với nhiều thư viện và công cụ bên thứ ba. Framework hỗ trợ SQLAlchemy, cho phép làm việc hiệu quả với cơ sở dữ liệu. Dependency injection system của FastAPI giúp tổ chức và tái sử dụng code một cách hiệu quả, đồng thời duy trì tính maintainable và testable của ứng dụng. Hệ thống này cho phép

developers định nghĩa dependencies một cách declarative và tự động inject chúng vào các endpoint functions.

#### **2.6.4 *Next.js Language***

Next.js là một React framework full-stack được tối ưu hóa cho production, được tạo ra bởi Vercel. Framework này cho phép xây dựng các ứng dụng web tương tác, động và nhanh chóng bằng cách sử dụng React Components cho giao diện người dùng và Next.js cho các tính năng bổ sung cùng tối ưu hóa [13].

##### **2.6.4.1 Server và Client Components**

Next.js phân biệt rõ ràng giữa Server Components và Client Components. Server Components được render trên server và không gửi JavaScript bổ sung đến client, giúp giảm bundle size và cải thiện hiệu suất. Ngược lại, Client Components được đánh dấu bằng directive 'use client' và cung cấp interactivity cùng truy cập browser APIs [14].

Quá trình rendering trong Next.js bao gồm việc tạo React Server Component Payload (RSC Payload) - một định dạng binary compact chứa kết quả render của Server Components và Client Components.

##### **2.6.4.2 Full-stack capabilities**

Next.js 15 mang đến khả năng full-stack mạnh mẽ với Server Actions, cho phép chạy server code bằng cách gọi function và dễ dàng revalidate cached data. Framework cũng cung cấp Route Handlers để xây dựng API endpoints, Middleware để kiểm soát incoming requests, và hỗ trợ đa dạng các phương thức styling bao gồm CSS Modules và Tailwind CSS.

#### **2.6.5 *Crow***

Crow là một micro-framework viết hoàn toàn bằng C++, được thiết kế để giúp các lập trình viên nhanh chóng xây dựng các RESTful API với hiệu năng cao, tận dụng sức mạnh của C++. Về cơ bản, Crow vận hành trên nền tảng C++14, sử dụng thư viện Boost.Asio để xử lý I/O bất đồng bộ. Nhờ kiến trúc non-blocking I/O, Crow có thể phục vụ hàng nghìn kết nối đồng thời một cách hiệu quả.

Điểm nổi bật của Crow nằm ở việc nó là thư viện header-only, nghĩa là người lập trình chỉ cần #include các file tiêu đề mà không phải biên dịch thêm thư viện riêng. Điều này loại bỏ hoàn toàn gánh nặng về quản lý thư viện phụ thuộc, cấu hình build system phức tạp. Đối với một ứng dụng C++ chuyên dụng như máy quét mạng, việc nhúng một web server nhẹ nhàng như Crow là lựa chọn lý tưởng, giúp giữ cho ứng dụng gọn nhẹ và dễ triển khai. Bên cạnh đó, Crow hỗ trợ chạy đa luồng để tận dụng tối đa CPU đa nhân. Thêm nữa, Crow cũng hỗ trợ SSL/TLS qua OpenSSL và khả năng mở rộng WebSocket khiến đây là framework rất phù hợp cho ứng dụng của đồ án yêu

cầu tính thời gian thực (hiển thị kết quả dò quét Wi-Fi và CSI).

#### **2.6.6 MongoDB**

MongoDB là một hệ quản trị cơ sở dữ liệu NoSQL (Not Only SQL) mã nguồn mở, thuộc loại hướng tài liệu (document-oriented). Thay vì lưu trữ dữ liệu trong các bảng với các hàng và cột có cấu trúc cố định như cơ sở dữ liệu quan hệ (SQL), MongoDB lưu trữ dữ liệu dưới dạng các tài liệu (documents). Các tài liệu này có cấu trúc tương tự như JSON (sử dụng một định dạng nhị phân gọi là BSON - Binary JSON), cho phép lưu trữ dữ liệu phân cấp và có cấu trúc linh hoạt.

#### **2.7 Nghiên cứu các sản phẩm/dự án tương tự**

#### **2.8 Lựa chọn công nghệ phù hợp**

#### **Kết luận chương**

## CHƯƠNG 3. THIẾT KẾ HỆ THỐNG

### Mở đầu chương

Trong chương này, chúng ta sẽ trình bày quá trình thiết kế hệ thống/sản phẩm từ giai đoạn ý tưởng ban đầu đến mô hình cụ thể. Việc thiết kế được thực hiện dựa trên việc phân tích kỹ lưỡng các yêu cầu chức năng và phi chức năng, đồng thời xây dựng mô hình tổng thể, thiết kế phần cứng và phần mềm, cũng như phương thức tích hợp các thành phần để đảm bảo hệ thống hoạt động hiệu quả.

### 3.1 Phần mềm quản lý

#### 3.1.1 Phân tích yêu cầu hệ thống/sản phẩm

##### 3.1.1.1 Yêu cầu kỹ thuật

Phần mềm quản lý thiết bị dò quét WiFi cần đáp ứng các yêu cầu kỹ thuật sau:

- Hiệu năng: Xử lý và lưu trữ dữ liệu từ nhiều thiết bị dò quét đồng thời.
- Khả năng mở rộng: Hệ thống có thể mở rộng để quản lý thêm thiết bị dò quét mà không ảnh hưởng đến hiệu suất tổng thể.
- Độ ổn định: Hoạt động liên tục 24/7, chịu được các sự cố mạng hoặc phần cứng nhỏ mà không làm gián đoạn dịch vụ.
- Băng thông: Hệ thống phải đáp ứng được lưu lượng dữ liệu lớn khi nhiều thiết bị đồng thời truyền dữ liệu về server.
- Thời gian phản hồi: Giao diện người dùng phản hồi nhanh chóng, đặc biệt khi hiển thị dữ liệu real-time và báo cáo.

##### 3.1.1.2 Yêu cầu chức năng

Các yêu cầu chức năng của phần mềm quản lý thiết bị được đưa ra dưới đây:

STT	Mô tả
<b>1</b>	<b>Quản lý sensor</b>
<b>1.1</b>	Hiển thị trạng thái các Sensor
<b>1.2</b>	Xem, thay đổi thông tin Sensor: ID, Name, Token, Position
<b>1.3</b>	Thêm mới/ Xóa Sensor
<b>2</b>	<b>Giám sát, điều khiển một Sensor cụ thể</b>
<b>2.1</b>	Quản lý Card: Xem danh sách, Cấu hình card
<b>2.2</b>	Quản lý kết quả quét: Hiển thị, Lọc, Sắp xếp, Export/Import
<b>3</b>	<b>Thông báo</b>
<b>3.1</b>	Cài đặt luật thông báo (trên giao diện, import file), Export file
<b>3.2</b>	Hiển thị thông báo: dạng danh sách (bấm vào từng thông báo), dạng icon, dạng danh sách thông báo
<b>4</b>	<b>Xem lịch sử</b>
<b>4.1</b>	Bộ lọc: chọn sensor, thời gian, lọc theo BSSID/ESSID, MAC, hoặc thông báo
<b>4.2</b>	Hiển thị kết quả lọc: danh sách AP, danh sách Client của AP, danh sách Client từ lọc Client
<b>5</b>	<b>Cơ sở dữ liệu</b>
<b>5.1</b>	Nhận liên tục và lưu trữ dữ liệu từ Sensor
<b>5.2</b>	Kiểm tra luật để tạo và lưu thông báo vào cơ sở dữ liệu
<b>5.3</b>	Đẩy thông báo tới Center Web khi có thông báo mới

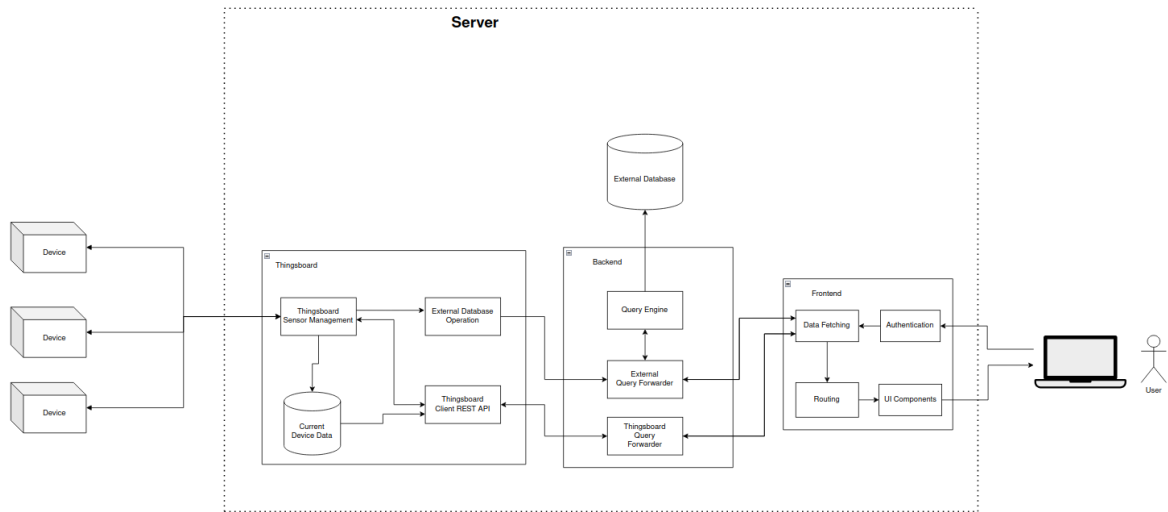
*Bảng 3.1. Yêu cầu chức năng của Hệ thống quản lý thiết bị*

#### **3.1.1.3 Yêu cầu phi chức năng**

- Bảo mật: Hệ thống phải đảm bảo an toàn thông tin, mã hóa dữ liệu, xác thực người dùng và thiết bị, hạn chế truy cập trái phép.
- Khả năng sử dụng: Giao diện thân thiện, dễ sử dụng, hỗ trợ nhiều thiết bị truy cập (máy tính, điện thoại, tablet).
- Khả năng bảo trì: Dễ dàng cập nhật, sửa lỗi, nâng cấp hệ thống mà không ảnh hưởng đến hoạt động hiện tại.

### 3.1.2 Mô hình thiết kế tổng thể

#### 3.1.2.1 Sơ đồ khối



Hình 3.1. Kiến trúc của Phần mềm quản lý thiết bị

#### 3.1.2.2 Nguyên lý hoạt động

Phần mềm quản lý thiết bị sử dụng ThingsBoard làm nền tảng trung gian để thu thập dữ liệu từ các thiết bị cảm biến, đồng thời tích hợp Backend để xử lý, lưu trữ vào cơ sở dữ liệu ngoài và phản hồi thông tin theo thời gian thực tới người dùng. Quy trình hoạt động chi tiết như sau:

##### 1. Thiết bị gửi dữ liệu:

- Các thiết bị đầu cuối (Raspberry Pi) thực hiện dò quét WiFi, sau đó gửi dữ liệu thu thập được đến hệ thống thông qua giao thức MQTT chuẩn.
- Mỗi thiết bị được cấu hình và định danh trong ThingsBoard để phục vụ việc quản lý và giám sát tập trung.

##### 2. Tiếp nhận và xử lý dữ liệu tại ThingsBoard:

- Dữ liệu được tiếp nhận bởi module ThingsBoard Sensor Management, nơi quản lý trạng thái, cấu hình và định danh thiết bị.
- Đồng thời, dữ liệu cũng được cập nhật vào Current Device Data, đóng vai trò như bộ nhớ đệm lưu trữ giá trị mới nhất.

##### 3. Lưu trữ vào cơ sở dữ liệu ngoài:

- Tất cả dữ liệu thu được từ thiết bị sẽ được chuyển tiếp từ ThingsBoard sang module External Database Operation.

- Tại đây, dữ liệu được xử lý và ghi vào External Database, đảm bảo lưu trữ dài hạn và phục vụ phân tích truy vấn lịch sử.

#### 4. Gửi dữ liệu thời gian thực đến người dùng:

- Ngay khi dữ liệu được cập nhật, hệ thống sẽ thông qua ThingsBoard Client REST API gửi dữ liệu mới đến phần Frontend
- Module Data Fetching tiếp nhận và hiển thị dữ liệu theo thời gian thực, đảm bảo người dùng quan sát liên tục hoạt động thiết bị.

#### 5. Truy vấn cơ sở dữ liệu ngoại và điều khiển:

- Người dùng tương tác với hệ thống qua giao diện web, gửi yêu cầu như xem lịch sử, lọc dữ liệu, điều khiển thiết bị...
- Các yêu cầu được chuyển tới Backend, nơi tích hợp:
  - ThingsBoard Query Forwarder: chuyển tiếp yêu cầu đến ThingsBoard khi cần truy vấn dữ liệu mới nhất.
  - External Query Forwarder: xử lý các truy vấn dữ liệu lịch sử từ External Database.
- Query Engine: sẽ xử lý logic và phản hồi kết quả về frontend.

#### 6. Xác thực và trình bày dữ liệu:

- Module Authentication đảm bảo chỉ người dùng hợp lệ mới được truy cập hệ thống.
- Sau xác thực, dữ liệu được định tuyến bởi Routing và hiển thị trực quan qua UI Components, bao gồm bảng dữ liệu...

#### 7. Điều khiển thiết bị từ xa:

- Khi người dùng thực hiện các thao tác điều khiển (ví dụ bật/tắt chức năng), lệnh được gửi từ frontend tới backend.
- Backend sử dụng ThingsBoard Client REST API để chuyển tiếp lệnh đến ThingsBoard.
- Sau đó ThingsBoard gửi lệnh đến thiết bị theo giao thức đã định nghĩa (MQTT hoặc HTTP), đảm bảo điều khiển thiết bị theo thời gian thực.

### 3.1.3 *Thiết kế phần mềm*

#### 3.1.3.1 Chi tiết các thành phần chính

##### a. Thingsboard

ThingsBoard đóng vai trò là nền tảng IoT trung tâm của hệ thống, chuyên xử lý dữ liệu cảm biến, lưu trữ tạm thời và quản lý thiết bị. Trong khối ThingsBoard có một số thành phần chính như sau:

- ThingsBoard Sensor Management: chịu trách nhiệm quản lý các thiết bị, giám sát trạng thái kết nối và là đầu vào của dữ liệu trước khi được chuyển tiếp tới nơi khác.
- Current Device Data: là cơ sở dữ liệu nội của Thingsboard (cơ sở dữ liệu mặc định), dùng để hiển thị dữ liệu thời gian thực tại ThingsBoard.
- External Database Operation: cung cấp API chuyển tiếp các dữ liệu cần lưu trữ lâu dài từ ThingsBoard sang cơ sở dữ liệu ngoại (PostgreSQL) thông qua Backend.
- ThingsBoard Client REST API: cung cấp API giúp backend có thể truy cập thông tin về thiết bị, dữ liệu hoặc trạng thái thiết bị hoặc tự Thingsboard chuyển tiếp dữ liệu tới Backend thông qua HTTP.

Thingsboard API gồm 2 loại chính: device API và server-side API, ở trong khuôn khổ Đồ Án Tốt Nghiệp, nhóm đã sử dụng 1 số API sau:

- MQTT API (Device API) MQTT là một giao thức nhắn tin xuất bản-đăng ký (publish-subscribe) nhẹ, làm cho nó trở thành lựa chọn lý tưởng cho nhiều thiết bị IoT. Các nút máy chủ ThingsBoard hoạt động như một MQTT Broker, hỗ trợ mức QoS (Quality of Service) 0 (tối đa một lần) và 1 (ít nhất một lần), cùng với một tập hợp các chủ đề (topics) có thể cấu hình. Các thiết bị xác thực bằng cách sử dụng một mã truy cập (\$ACCESS\_TOKEN) trong trường tên người dùng (username) của tin nhắn MQTT CONNECT. Các chức năng MQTT Device API chính đã sử dụng:

– MQTT Connect:

- \* Mục đích: Thiết lập kết nối giữa thiết bị và máy chủ MQTT của ThingsBoard.
- \* Xác thực: Các thiết bị gửi tin nhắn MQTT CONNECT với \$ACCESS\_TOKEN trong trường tên người dùng.
- \* Mã trả về:
  - 0x00 Connected: Kết nối thành công với máy chủ MQTT của ThingsBoard.
  - 0x04 Connection Refused, bad username or password: Tên người dùng trống.
  - 0x05 Connection Refused, not authorized: \$ACCESS\_TOKEN không hợp lệ.



- \* Xác thực thay thế: Có thể sử dụng Chứng chỉ X.509 hoặc Thông tin đăng nhập MQTT cơ bản (kết hợp Client ID, tên người dùng và mật khẩu).

#### – Attributes API:

- \* Mục đích: Quản lý các thuộc tính của thiết bị (client-side và shared) trên máy chủ ThingsBoard.
- \* Xuất bản cập nhật thuộc tính lên máy chủ:
  - Mục đích: Tải lên các thuộc tính client-side của thiết bị.
  - Chủ đề (Topic): `v1/devices/me/attributes`
  - Định dạng dữ liệu: `{"attribute1": "value1", "attribute2": true}`
- \* Yêu cầu giá trị thuộc tính từ máy chủ:
  - Mục đích: Yêu cầu các thuộc tính client-side hoặc shared của thiết bị.
  - Chủ đề yêu cầu: `v1/devices/me/attributes/request/$request_id`
  - Chủ đề đăng ký: `v1/devices/me/attributes/response/+`
- \* Đăng ký nhận cập nhật thuộc tính từ máy chủ:
  - Mục đích: Nhận các thay đổi thuộc tính shared của thiết bị được khởi tạo bởi các thành phần phía máy chủ.
  - Chủ đề đăng ký: `v1/devices/me/attributes`
  - Định dạng cập nhật: `{"key1": "value1"}`

#### – RPC API:

- \* Mục đích: Cho phép các lệnh gọi thủ tục từ xa (RPC) giữa máy chủ và thiết bị.
- \* Server-side RPC (Lệnh từ máy chủ gửi đến thiết bị):
  - Chủ đề đăng ký: `v1/devices/me/rpc/request/+`
  - Chủ đề phản hồi: `v1/devices/me/rpc/response/$request_id`
- \* Client-side RPC (Lệnh từ thiết bị gửi đến máy chủ):
  - Chủ đề yêu cầu: `v1/devices/me/rpc/request/$request_id`
  - Chủ đề đăng ký: `v1/devices/me/rpc/response/$request_id`

#### • Administration REST API (Server-side API)

- Mục đích: Cho phép quản lý hệ thống ThingsBoard từ phía máy chủ, bao gồm người dùng, thiết bị, dashboard, v.v.
- Xác thực: Sử dụng JWT (JSON Web Token) để xác thực sau khi đăng nhập qua endpoint `/api/auth/login`.
- Các ví dụ endpoint:
  - \* `POST /api/device`: Tạo thiết bị mới.
  - \* `GET /api/user`: Lấy thông tin người dùng.

- \* DELETE /api/device/{deviceId}: Xóa thiết bị theo ID.

- Định dạng dữ liệu: JSON, dùng các phương thức HTTP như GET, POST, PUT, DELETE.

- REST API Call Node (Rule Chain)

- Mục đích: Cho phép gửi dữ liệu hoặc sự kiện từ Rule Chain ra bên ngoài thông qua các yêu cầu HTTP (REST API).

- Chức năng chính:

- \* Gửi dữ liệu telemetry, attributes hoặc sự kiện đến các dịch vụ hoặc hệ thống bên ngoài.

- \* Hỗ trợ cấu hình endpoint URL, phương thức HTTP (GET, POST, PUT, DELETE), tiêu đề (headers) và payload.

- \* Cho phép xử lý và định dạng dữ liệu gửi đi theo yêu cầu.

- Định dạng dữ liệu: Hỗ trợ gửi dữ liệu ở dạng JSON hoặc các định dạng khác theo cấu hình.

## **b. Backend**

Backend là thành phần trung gian, kết nối giữa Frontend và Thingsboard hoặc Database ngoại. Backend được xây dựng bằng FastAPI, đảm nhận việc xử lý logic nghiệp vụ, xác thực người dùng, chuyển tiếp truy vấn và xử lý dữ liệu. Các thành phần chính trong backend gồm:

- External Query Forwarder: đóng vai trò tiếp nhận các truy vấn từ Frontend đến External Database thông qua Query Engine hoặc đẩy dữ liệu thu được từ Thingsboard đến External Database để lưu trữ lâu dài.

- ThingsBoard Query Forwarder: xử lý các yêu cầu trao đổi giữa Frontend và Thingsboard thông qua REST API.

- Query Engine: là bộ xử lý trung tâm các truy vấn trực tiếp đến External Database.

Theo yêu cầu chức năng của Phần mềm quản lý, thiết kế những API phù hợp với các chức năng;

- Thiết kế API quản lý sensor

- API lấy danh sách tất cả các sensor: Lấy danh sách các Device phù hợp từ Thingsboard và trả về thông tin tất cả Device (tên, ID, vị trí đặt, trạng thái)

- API thêm sensor mới: Cho phép đăng ký Device mới từ Thingsboard các thông số cơ bản (tên, vị trí đặt)

- API cập nhật thông tin sensor theo ID: Sửa thông tin Device từ Thingsboard thông qua ID định danh. Bao gồm các thông tin sau: đổi tên, vị trí đặt; cấu hình thông báo (cảnh báo về các địa chỉ MAC mà Device quét được có nằm trong danh sách cảnh báo hay không)
- API xóa sensor theo ID: Xóa bỏ Device khỏi Thingsboard thông qua ID định danh.
- Thiết kế API quản lý card
  - API danh sách card của sensor: Hiển thị thông tin các card mạng trên Device cụ thể (MAC address, trạng thái kết nối, kênh quét...)
  - API cập nhật thông tin card của sensor: Bật/tắt chế độ monitor, chọn kênh quét.
- Thiết kế API điều khiển bật, tắt xem kết quả quét
  - API bật quét kết quả: Khởi tạo tiến trình quét từ các card từ Device cụ thể.
  - API dừng quét kết quả: Dừng tiến trình quét đang thực thi và xuất báo cáo kết quả quét.
- Thiết kế API truy vấn cơ sở dữ liệu
  - API truy vấn danh sách AP: Liệt kê các Access Point quét được từ Database (danh sách hiển thị với mỗi AP chỉ hiển thị 1 dòng duy nhất).
  - API truy vấn danh sách STA: Liệt kê các Station quét được từ Database.
  - API truy vấn STA của 1 AP tại 1 thời điểm: Liệt kê các Station đang kết nối với Access Point cụ thể tại 1 thời điểm từ Database.
  - API truy vấn danh sách notification: Cung cấp các log cảnh báo cấu hình cho các địa chỉ MAC từ Database.
  - API truy vấn lịch sử các AP: Hiển thị lịch sử quét được các Access Point tương ứng từ Database (khác với API truy vấn danh sách AP ở chỗ là danh sách hiển thị ở dạng timestamp).
  - API truy vấn lịch sử các STA: Hiển thị lịch sử quét được các Statuon tương ứng từ Database.
- Thiết kế API nhận dữ liệu thời gian thực
  - API gửi kết quả quét: Lấy kết quả quét nhận được từ Device thông qua Thingsboard.
  - API đồng bộ thông tin sensor: Cập nhật trạng thái online/offline của từng Device trong Thingsboard.

- API gửi thông báo: Cập nhật thông báo liên tục khi được cấu hình.
- Thiết kế API cập nhật cơ sở dữ liệu
  - API Cập nhật metadata của sensor trong DB: Cập nhật thông tin sensor được thay đổi vào trong cơ sở dữ liệu.
  - API Lưu trữ data từ sensor vào DB: Lưu trữ dữ liệu quét được vào cơ sở dữ liệu để lưu trữ lâu dài.
- Thiết kế API quản lý người dùng
  - API xác thực đăng nhập người dùng: Cơ chế xác thực password bằng bcrypt để so sánh thông tin người dùng từ Database với thông tin nhập từ giao diện (xác thực với JWT có timeout).
  - API cập nhật thông tin người dùng: Cập nhật thông tin người dùng (username, password, email, name) với xác thực lại JWT.

#### **c. Cơ sở dữ liệu ngoài (External Database)**

Hệ thống sử dụng một cơ sở dữ liệu PostgreSQL truy cập bởi backend thông qua Query Engine, đảm bảo dữ liệu có thể được truy vấn các trường dữ liệu theo yêu cầu. Cơ sở dữ liệu thiết kế gồm các bảng sau:

- Bảng notislist: chứa danh sách các thông báo/cảnh báo theo địa chỉ MAC đã cấu hình.
- Bảng applist: chứa thông tin của các Access Point quét được từ Device.
- Bảng staslist: chứa thông tin của các Station quét được từ Device.
- Bảng record: chứa metadata cho mỗi đợt quét (sử dụng UUID làm khóa chính để nhóm các bản ghi từ applist và staslist theo từng phiên quét).
- Bảng users: chứa thông tin người dùng.

#### **d. Frontend**

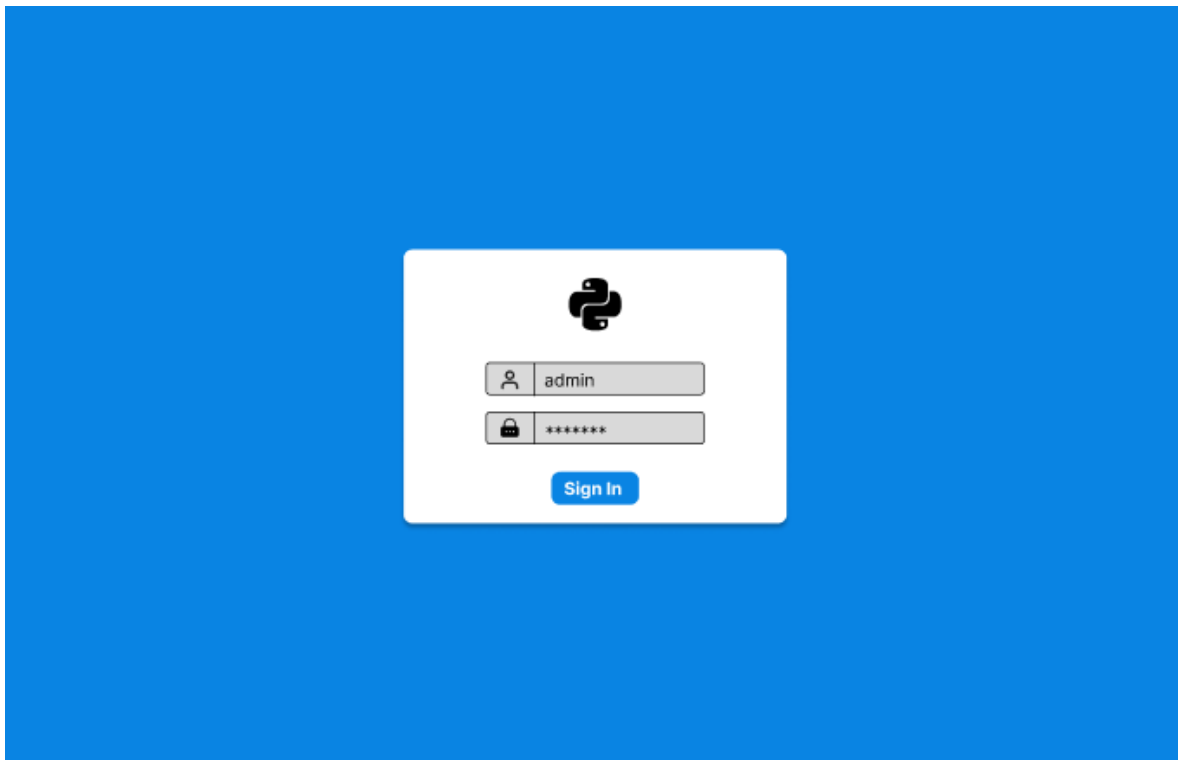
Frontend là giao diện người dùng cuối. Nó cho phép người dùng đăng nhập vào hệ thống, theo dõi thông tin cảm biến, và gửi các yêu cầu điều khiển đến thiết bị. Frontend giao tiếp với backend thông qua HTTP API. Mỗi thao tác của người dùng như yêu cầu thay đổi trạng thái thiết bị đều được frontend chuyển thành yêu cầu API phù hợp gửi đến backend, sau đó backend sẽ xử lý và chuyển lệnh đến thiết bị thông qua ThingsBoard.

Các module chức năng chính trong frontend của đề án được tổ chức thành nhiều thành phần riêng biệt, mỗi thành phần đảm nhiệm một vai trò cụ thể nhằm đảm bảo tính mạch lạc, khả năng mở rộng và dễ bảo trì cho toàn bộ ứng dụng web:

- Authentication: Xác thực người dùng (đăng nhập/đăng xuất) Đảm bảo chỉ người dùng hợp lệ mới truy cập được vào các chức năng quan trọng như dashboard, quản lý thiết bị (có thể dùng JWT).
- Data Fetching: Giao tiếp với backend API để:
  - Lấy dữ liệu quét được từ sensor, cards (MAC, Channel, RSSI,...).
  - Lấy trạng thái sensor, cards (online/offline).
  - Lấy dữ liệu lịch sử truy vấn từ cơ sở dữ liệu ngoài.
  - Gửi cấu hình xuống sensor, cards.
- Routing: quản lý các tuyến đường trong ứng dụng (/login, /dashboard, /history) để liên kết dữ liệu nhận được tới các UI Components
- UI Components: Bao gồm các thành phần hiển thị và tương tác:
  - Bảng: hiện danh sách sensor, card, kết quả quét được,...
  - Nút điều khiển: bật/tắt, gửi lệnh.
  - Biểu mẫu: nhập thông tin sensor, bộ lọc tìm kiếm,...

### **Thiết kế giao diện hiển thị trên Figma**

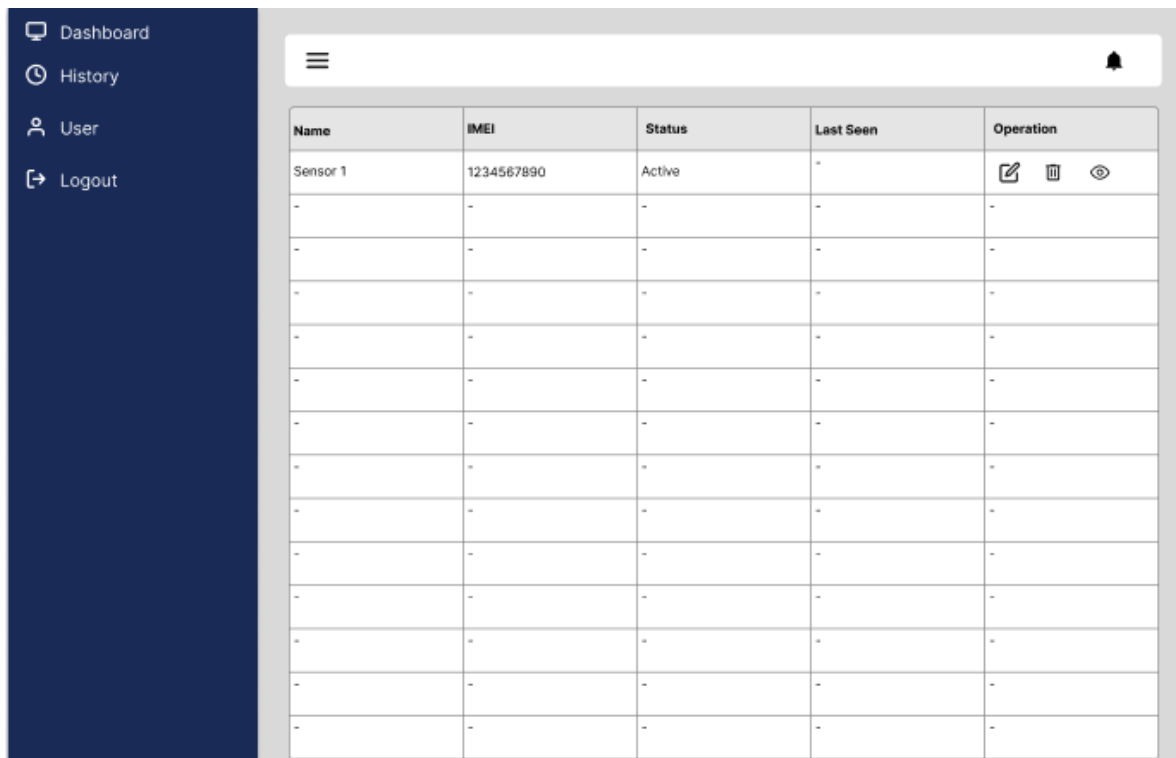
- Trang Login  
 Khi người dùng truy cập vào hệ thống, trang đầu tiên hiển thị sẽ luôn là trang đăng nhập. Đây là cơ chế bắt buộc nhằm đảm bảo an toàn và kiểm soát truy cập. Người dùng không thể xem hoặc truy cập bất kỳ trang nào khác nếu chưa thực hiện đăng nhập hợp lệ.



*Hình 3.2. Trang Login*

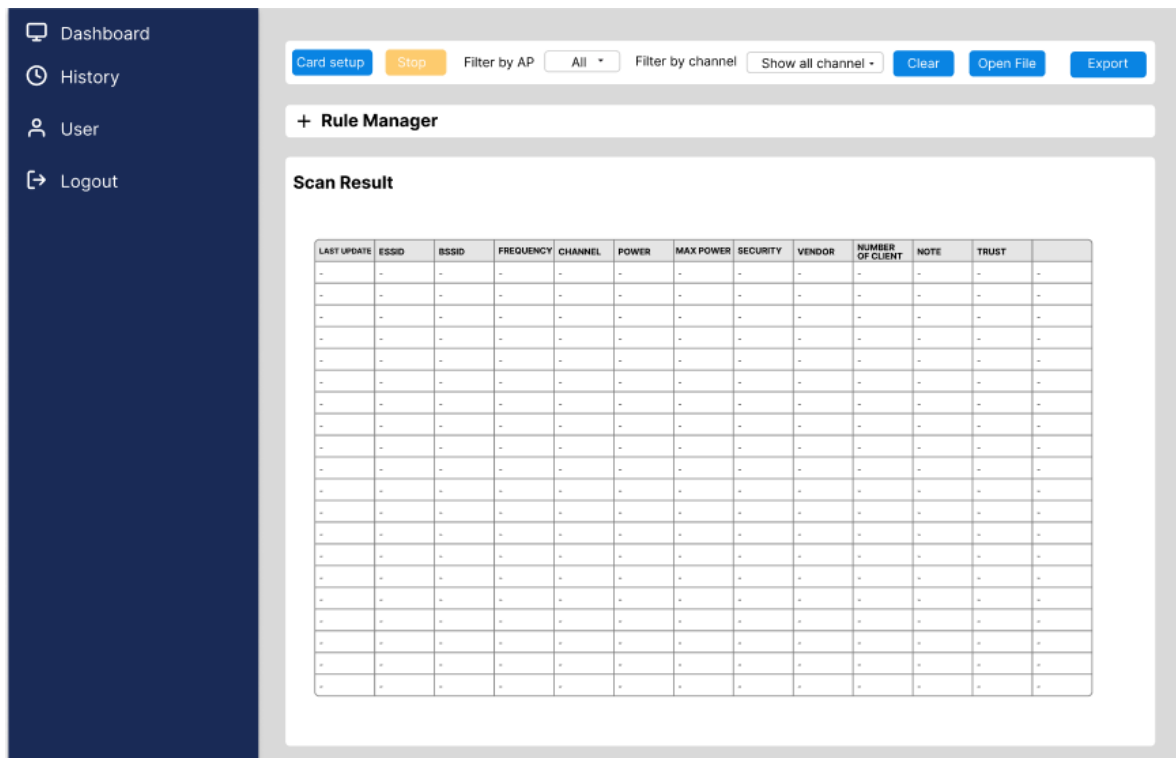
- Trang đăng nhập yêu cầu người dùng nhập thông tin xác thực gồm:
  - \* Tên đăng nhập (username hoặc email)
  - \* Mật khẩu
- Xác thực thành công:
  - \* Người dùng sẽ được chuyển đến giao diện chính phù hợp.
  - \* Hệ thống cấp phiên làm việc (*session*) hoặc *token* để xác nhận người dùng đã đăng nhập.
- Nếu chưa đăng nhập hoặc đăng nhập sai:
  - \* Người dùng sẽ không thể truy cập các trang nội bộ như trang quản lý, bảng điều khiển, báo cáo, v.v.
  - \* Hệ thống có thể tự động chuyển hướng về trang đăng nhập khi phát hiện truy cập trái phép.
- Trang Dashboard

Trang Dashboard là giao diện trung tâm của hệ thống, dùng để hiển thị, theo dõi và quản lý các sensor (cảm biến) đang được giám sát. Mỗi sensor được trình bày dưới dạng một hàng trong bảng dữ liệu, với đầy đủ thông tin nhận diện và trạng thái hoạt động.



Hình 3.3. Trang Dashboard

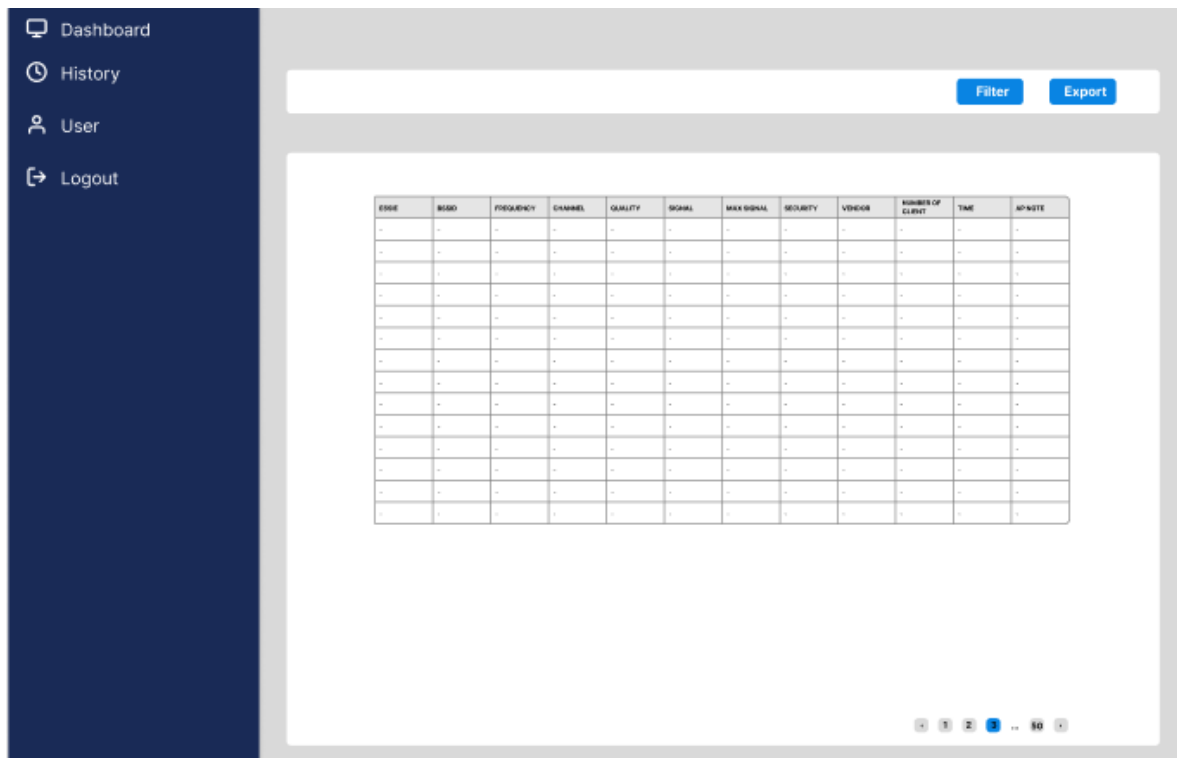
- Hiển thị danh sách sensor: mỗi sensor phù hợp được hiển thị trên mỗi dòng của bảng.
  - Xem chi tiết sensor: mở giao diện cho phép xem thông tin chi tiết của sensor đó: danh sách cards, kết quả quét được,...
  - Sửa thông tin sensor: mở giao diện chỉnh sửa thông tin như tên thiết bị, vị trí đặt, cấu hình notification...
  - Xóa sensor: xóa sensor khỏi hệ thống sau khi xác nhận.
  - Thêm sensor mới: cho phép nhập thông tin thiết bị mới để thêm vào hệ thống theo dõi (nằm ở phần trên hoặc góc phải của bảng).
- Trang Sensor Details  
Trang hiển thị thông tin của 1 sensor chi tiết gồm các thông tin card của sensor và kết quả quét được từ sensor đó.



Hình 3.4. Trang Sensor Details

- Nút Card Setup: Hiển thị thông tin các card wifi của sensor, có thể cấu hình các card (bật tắt card và chọn channel để quét).
  - Nút Start/Stop: Bật, tắt hiển thị kết quả quét được tại bảng (hiển thị thời gian thực).
  - Bộ lọc theo AP và Channel: Lọc theo AP và Channel muốn hiển thị tại bảng.
  - Bảng Scan Result: Chứa thông tin quét được theo thời gian thực của sensor
  - Các nút bổ sung: Một vài chức năng tùy chỉnh như nhập, xuất và xóa file.
- Trang History  
Trang hiển thị lịch sử dữ liệu lịch sử quét từ các thiết bị được lưu trữ ở cơ sở dữ liệu ngoài.

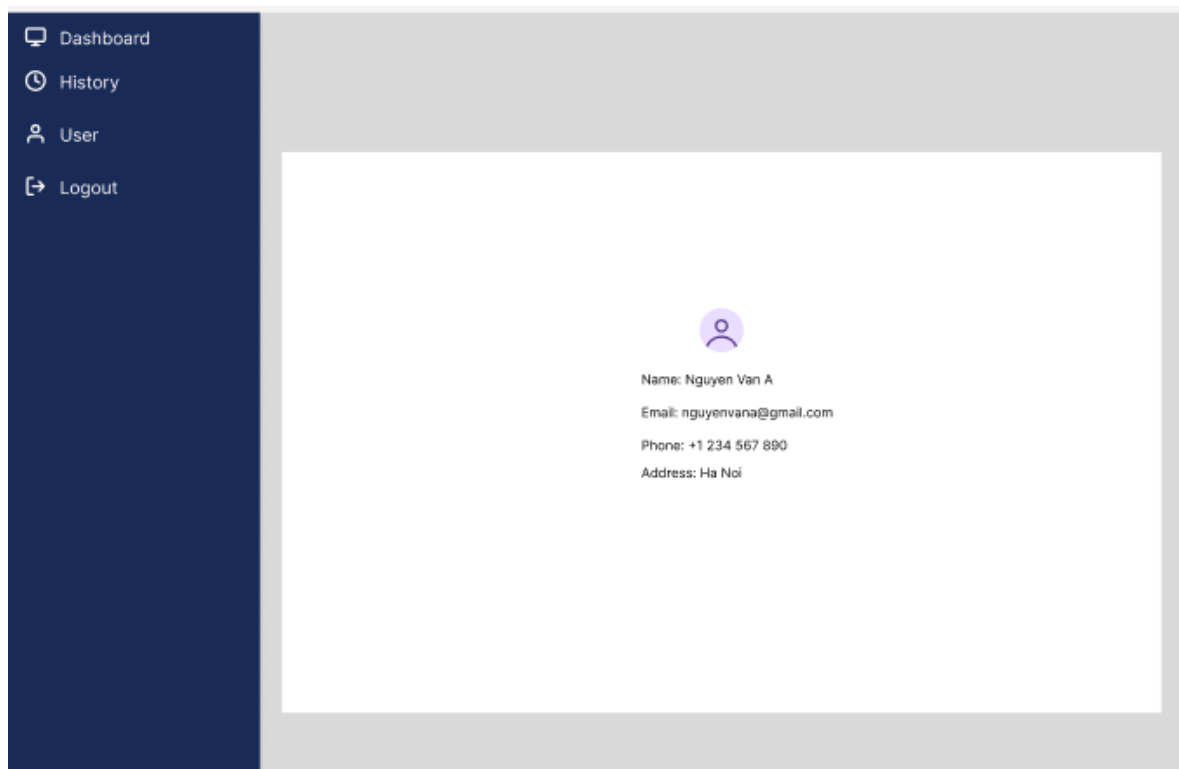




Hình 3.5. Trang History

- Bảng dữ liệu lịch sử: dữ liệu quét được từ cơ sở dữ liệu ngoài
  - Nút Filter: Cho phép lọc dữ liệu theo các tiêu chí: MAC của STA, BSSID, ESSID, notification...
  - Nút Export: Cho phép xuất dữ liệu ra file CSV để phân tích bên ngoài.
  - Phân trang: Cho phép duyệt qua nhiều trang dữ liệu khi có số lượng lớn bản ghi lịch sử.
- Trang User
 

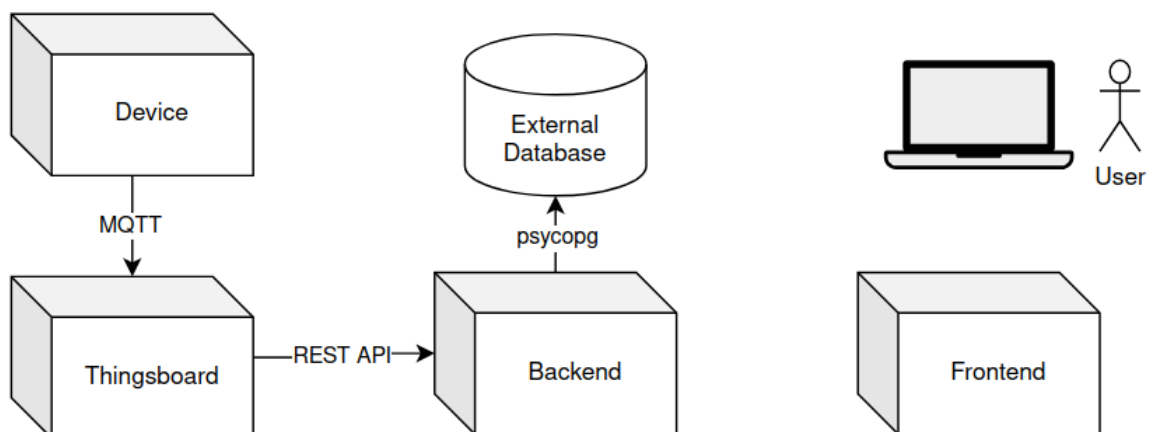
Trang User được sử dụng để hiển thị các thông tin cá nhân của người dùng hiện đang đăng nhập.



Hình 3.6. Trang User

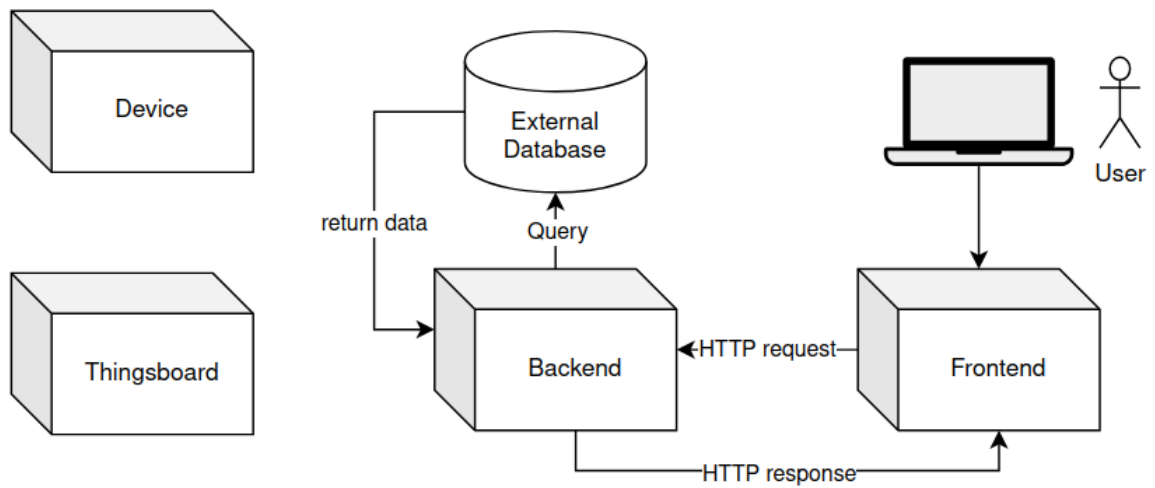
### 3.1.3.2 Phương thức kết nối

#### a. Lưu trữ dữ liệu vào cơ sở dữ liệu ngoại



Hình 3.7. Lưu trữ dữ liệu vào cơ sở dữ liệu ngoại

#### b. Truy vấn dữ liệu từ cơ sở dữ liệu ngoại

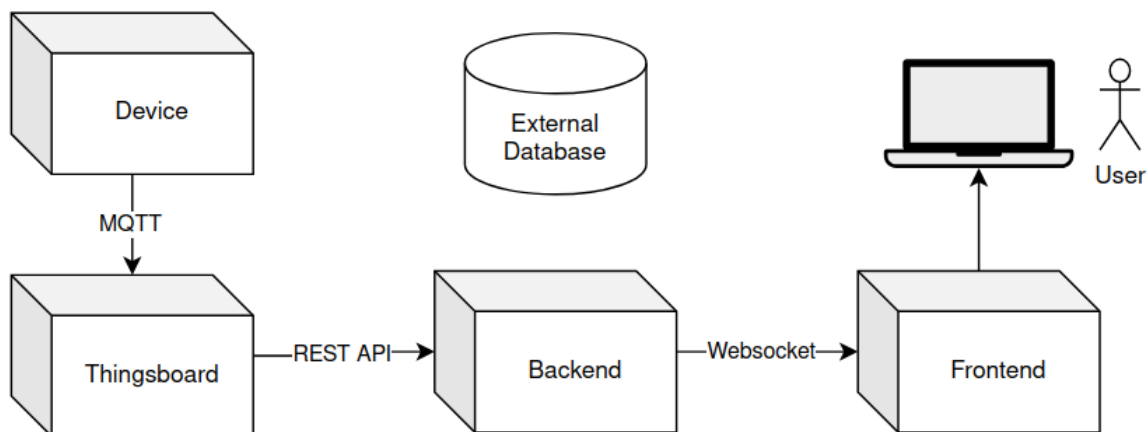


*Hình 3.8. Truy vấn dữ liệu từ cơ sở dữ liệu ngoại*

Sơ đồ trên phục vụ cho các API của Backend sau:

- API truy vấn danh sách AP.
- API truy vấn danh sách STA.
- API truy vấn STA của 1 AP tại 1 thời điểm.
- API truy vấn danh sách notification.
- API truy vấn lịch sử các AP.
- API truy vấn lịch sử các STA.
- API xác thực đăng nhập người dùng.
- API cập nhật thông tin người dùng.

c. Cập nhật dữ liệu thời gian thực

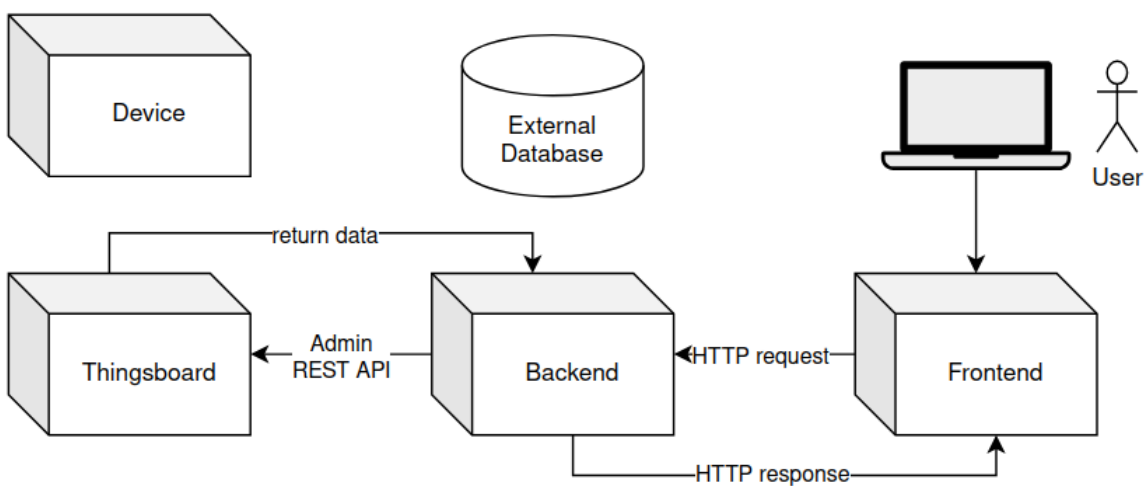


Hình 3.9. Cập nhật dữ liệu thời gian thực

Sơ đồ trên phục vụ cho các API của Backend sau:

- API gửi kết quả quét.
- API đồng bộ thông tin sensor.
- API gửi thông báo.

d. Cập nhật thông tin thiết bị



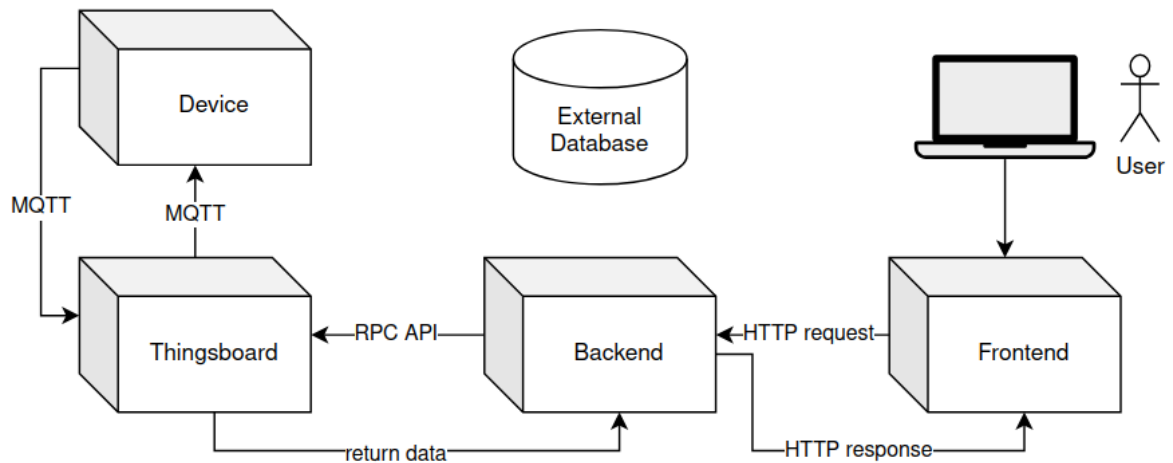
Hình 3.10. Cập nhật thông tin thiết bị

Sơ đồ trên phục vụ cho các API của Backend sau:

- API lấy danh sách tất cả các sensor.

- API thêm sensor mới.
- API cập nhật thông tin sensor theo ID.
- API xóa sensor theo ID.
- API danh sách card của sensor.

e. Điều khiển thiết bị từ xa



Hình 3.11. Điều khiển thiết bị từ xa

Sơ đồ trên phục vụ cho các API của Backend sau:

- API cập nhật thông tin card của sensor.
- API bật quét kết quả.
- API dừng quét kết quả.
- API Cập nhật metadata của sensor trong DB.

## 3.2 Thiết bị dò quét

### 3.2.1 Phân tích yêu cầu hệ thống/sản phẩm

**3.2.1.1 Yêu cầu kỹ thuật** Trình bày các yêu cầu kỹ thuật cần đáp ứng, ví dụ như hiệu năng, khả năng mở rộng, độ ổn định, băng thông, thời gian phản hồi,...

STT	Mô tả
<b>1</b>	<b>Quét và hiển thị điểm truy cập (Access Points - AP) và thiết bị (Stations)</b>
<b>1.1</b>	Quét và hiển thị các AP (cả AP ẩn) và Station đang kết nối tới các AP đó
<b>1.2</b>	Sắp xếp, lọc kết quả hiển thị theo kênh
<b>1.3</b>	Nhập/Xuất tập tin Excel chứa kết quả quét
<b>1.4</b>	Lưu kết quả quét vào cơ sở dữ liệu cục bộ
<b>2</b>	<b>Quản lý Card</b>
<b>2.1</b>	Hiển thị danh sách các Wi-Fi Card hiện có trên thiết bị
<b>2.2</b>	Cấu hình thủ công các Wi-Fi Card: Thay đổi kênh, chế độ hoạt động
<b>2.3</b>	Nhập/Xuất tập tin Excel chứa cấu hình Wi-Fi card
<b>3</b>	<b>Quản lý tài nguyên</b>
<b>3.1</b>	Nhập dữ liệu OUI (Organizationally Unique Identifier) cho việc nhận diện nhà sản xuất
<b>4</b>	<b>Đồng bộ</b>
<b>4.1</b>	Đẩy dữ liệu tới hệ thống trung tâm khi có kết nối
<b>4.2</b>	Không cho phép thao tác trên thiết bị khi có kết nối tới hệ thống trung tâm
<b>5</b>	<b>CSI</b>
<b>5.1</b>	Lấy & hiển thị dữ liệu CSI từ Card Wi-Fi Intel AX200/AX210
<b>5.2</b>	Tùy chọn lấy dữ liệu CSI theo kênh & định dạng lớp vật lý của khung Wi-Fi
<b>5.3</b>	Lưu dữ liệu CSI vào một tập tin và cơ sở dữ liệu cục bộ

*Bảng 3.2. Yêu cầu chức năng của phần mềm chạy trên thiết bị*

#### **3.2.1.2 Yêu cầu chức năng**

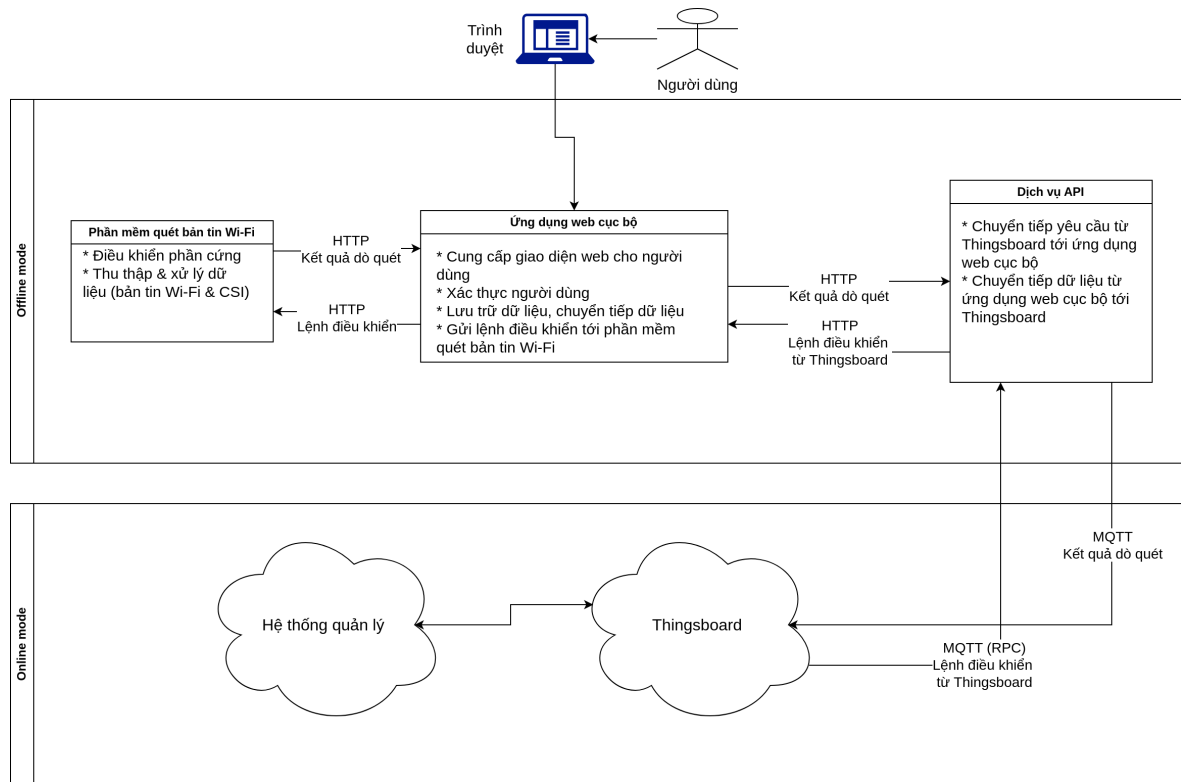
**3.2.1.3 Yêu cầu phi chức năng** Bao gồm các yêu cầu về bảo mật, độ tin cậy, khả năng sử dụng, khả năng bảo trì, tương thích,...

### **3.2.2 Mô hình thiết kế tổng thể**

#### **3.2.2.1 Sơ đồ khối**

#### **3.2.2.2 Nguyên lý hoạt động**

Phần mềm chạy trên thiết bị gồm ba mô-đun chính: phần mềm quét bản tin Wi-Fi, ứng dụng Web cục bộ, và dịch vụ API. Ba mô-đun này được triển khai ngay



Hình 3.12. Kiến trúc phần mềm chạy trên thiết bị

trên thiết bị, có thể hoạt động độc lập khi không thể kết nối tới hệ thống quản lý.

### 1. Phần mềm quét bản tin Wi-Fi:

Đây chính là mô-đun quan trọng nhất của hệ thống, một ứng dụng C++ hiệu suất cao hoạt động trên môi trường Linux. Mô-đun này gồm 2 thành phần. Thành phần thứ nhất liên tục thu thập các khung (frame) 802.11 thô, xử lý chúng để trích xuất thông tin có ý nghĩa, điều khiển phần cứng và cung cấp các API để các mô-đun còn lại có thể sử dụng ứng dụng này. Thành phần thứ hai thu thập & xử lý dữ liệu CSI, lưu trữ dữ liệu CSI và gửi dữ liệu CSI đến ứng dụng Web cục bộ để hiển thị. Cụ thể như sau:

- Thu thập khung dữ liệu (data frame) và khung quảng bá (beacon frame) bằng libpcap:
  - Phần mềm tận dụng thư viện libpcap để thực hiện việc thu thập khung bản tin Wi-Fi thô trên một hoặc nhiều card Wi-Fi. Card mạng không dây được đặt ở chế độ giám sát (monitor mode). Chế độ đặc biệt này cho phép card thu thập tất cả lưu lượng Wi-Fi trên một kênh nhất định, chứ không chỉ các khung bản tin được gửi đến thiết bị.
- Điều khiển phần cứng qua Netlink:
  - Mô-đun có thể thay đổi kênh Wi-Fi của một hoặc nhiều card Wi-Fi, cho phép hệ thống quét toàn bộ dải tần mà card Wi-Fi hỗ trợ.

- Mô-đun có khả năng lấy được thông tin của card Wi-Fi (dải tần card Wi-Fi hỗ trợ, tên, kênh hiện tại,...)
- CSI:
  - Thu thập giá trị CSI từ card Wi-Fi Intel AX200/AX210.
  - Lưu dữ liệu CSI vào cơ sở dữ liệu cục bộ theo thời gian thực. Gửi dữ liệu CSI tới Frontend để hiển thị.
  - Cho phép lưu dữ liệu CSI theo kênh, theo định dạng vật lý của khung bản tin.
- Cung cấp API cho các mô-đun khác:
  - Mô-đun cung cấp một bộ các RESTful API, cho phép các thành phần còn lại trong hệ thống tương tác và sử dụng các chức năng của mô-đun này.

## 2. Ứng dụng Web cục bộ:

Thành phần chính thứ hai của hệ thống là một ứng dụng web Python được xây dựng bằng framework Flask. Ứng dụng này đóng vai trò là giao diện chính cho người dùng và quản lý dữ liệu do phần mềm C++ thu thập. Cụ thể như sau:

- Thu thập và lưu trữ dữ liệu:
  - Ứng dụng Flask giao tiếp với phần mềm quét bản tin Wi-Fi để lấy dữ liệu Wi-Fi đã được xử lý. Điều này được thực hiện thông qua một cơ chế thăm dò (polling), trong đó ứng dụng Python định kỳ yêu cầu thông tin mới nhất từ kho dữ liệu trong bộ nhớ của ứng dụng C++.
- Hai chế độ riêng biệt để xử lý dữ liệu:
  - Chế độ ngoại tuyến (Offline Mode): Ở chế độ này, ứng dụng Flask lưu trữ dữ liệu Wi-Fi thu thập được trong một cơ sở dữ liệu MongoDB.
  - Chế độ trực tuyến (Online Mode): Ứng dụng hoạt động ở chế độ trực tuyến khi nó có kết nối với hệ thống quản lý. Khi đó, ứng dụng chuyển tiếp dữ liệu Wi-Fi nhận được (thời gian thực và những dữ liệu cũ đã lưu trong cơ sở dữ liệu) đến dịch vụ API thông qua các yêu cầu HTTP. Điều này cho phép tổng hợp và phân tích dữ liệu tập trung từ nhiều đơn vị quét phân tán.
- Tương tác và quản lý người dùng:
  - Ứng dụng Flask cung cấp một giao diện web cho phép người dùng tương tác và trực quan hóa dữ liệu Wi-Fi. Giao diện này hiển thị danh sách các mạng được phát hiện, các máy khách kết nối với mỗi mạng và cường độ tín hiệu tương ứng của chúng.



- Ứng dụng còn tích hợp một hệ thống quản lý người dùng. Tính năng này cho phép kiểm soát quyền truy cập vào hệ thống.

### 3. Dịch vụ API:

Đây là một ứng dụng được xây dựng bằng framework FastAPI, đóng vai trò là một cầu nối quan trọng giữa ứng dụng Web cục bộ và nền tảng IoT ThingsBoard. Chức năng cốt lõi của nó là chuyển đổi giao thức truyền thông, đảm bảo luồng dữ liệu được thông suốt theo cả hai chiều. Nó chuyển đổi các yêu cầu HTTP từ ứng dụng Web cục bộ thành tin nhắn MQTT cho ThingsBoard và chuyển đổi các lệnh gọi thủ tục từ xa (Remote Procedure Call - RPC) qua MQTT từ ThingsBoard thành các yêu cầu HTTP cho ứng dụng Web cục bộ. Cụ thể như sau:

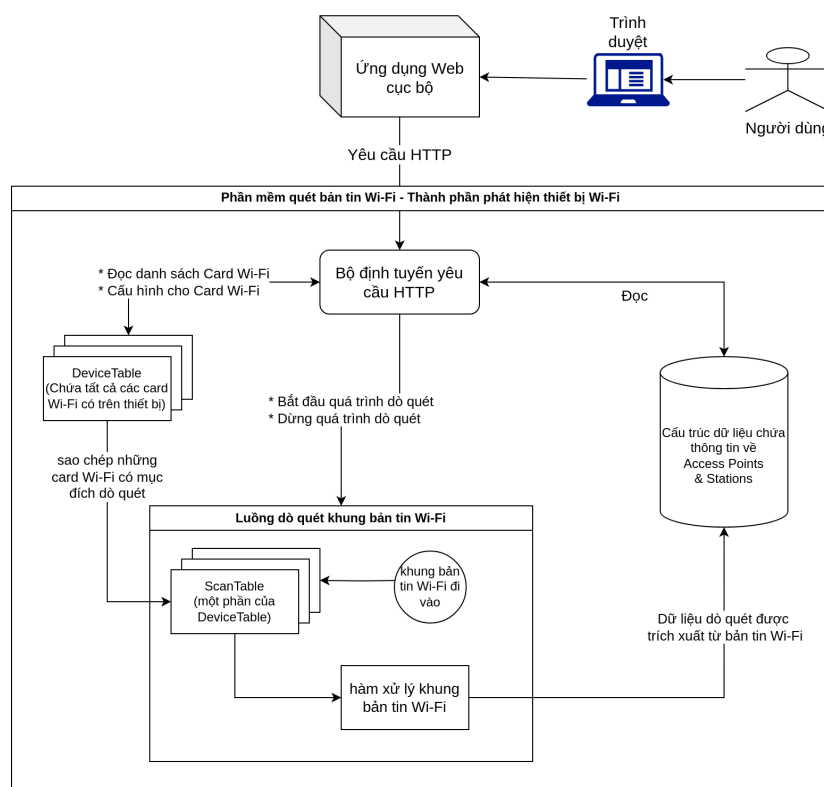
- Chuyển tiếp dữ liệu đến ThingsBoard (HTTP sang MQTT):
  - Dịch vụ API duy trì một kết nối máy khách liên tục đến MQTT broker của ThingsBoard.
  - Dịch vụ API cung cấp một URL. Ứng dụng Web cục bộ, khi ở chế độ trực tuyến, sẽ gửi dữ liệu Wi-Fi mà nó đã thu thập từ phần mềm quét bản tin Wi-Fi đến URL này bằng một yêu cầu HTTP POST.
  - Sau đó, dữ liệu Wi-Fi được chuyển tiếp tới ThingsBoard dưới dạng một tin nhắn MQTT. Dữ liệu của thiết bị được gửi đến nền tảng ThingsBoard, nơi hệ thống quản lý sẽ tiếp tục xử lý, lưu trữ, trực quan hóa.
- Xử lý các lệnh từ xa từ ThingsBoard (MQTT sang HTTP):
  - Dịch vụ API đăng ký (subscribe) một chủ đề (topic) MQTT cụ thể của ThingsBoard dành riêng cho Lệnh gọi thủ tục từ xa (RPC) cho một thiết bị nhất định. Điều này cho phép dịch vụ API lắng nghe các lệnh được gửi từ Thingsboard.
  - Khi người dùng kích hoạt một hành động trong giao diện người dùng của hệ thống quản lý (như nhấn nút để thay đổi kênh Wi-Fi), ThingsBoard sẽ phát hành một tin nhắn yêu cầu RPC đến chủ đề MQTT đó.
  - Dịch vụ API nhận được tin nhắn MQTT này. Nó phân tích lệnh và các tham số của nó từ gói tin trong tin nhắn. Sau đó, nó chuyển đổi lệnh này thành một yêu cầu HTTP và gửi nó đến ứng dụng Web cục bộ. Ứng dụng Web cục bộ sau đó sẽ xử lý yêu cầu này và tương tác với phần mềm C++ bên dưới để thực thi lệnh, chẳng hạn như thay đổi kênh của card Wi-Fi.

### 3.2.3 *Thiết kế phần mềm*

#### 3.2.3.1 Phần mềm quét bản tin Wi-Fi

- a. Xử lý khung quảng bá & khung dữ liệu (thành phần thứ nhất)

Thành phần này được thiết kế với kiến trúc được mô tả ở hình 3.13. nhằm mục đích dò quét và phân tích các gói tin Wi-Fi thô để phát hiện các thiết bị đang hoạt động trong mạng Wi-Fi xung quanh, bao gồm cả các điểm truy cập và các trạm. Người dùng thông qua một trình duyệt web, gửi các yêu cầu HTTP đến một ứng dụng web cục bộ. Ứng dụng này đóng vai trò trung gian, tiếp nhận yêu cầu và chuyển tiếp đến thành phần này.



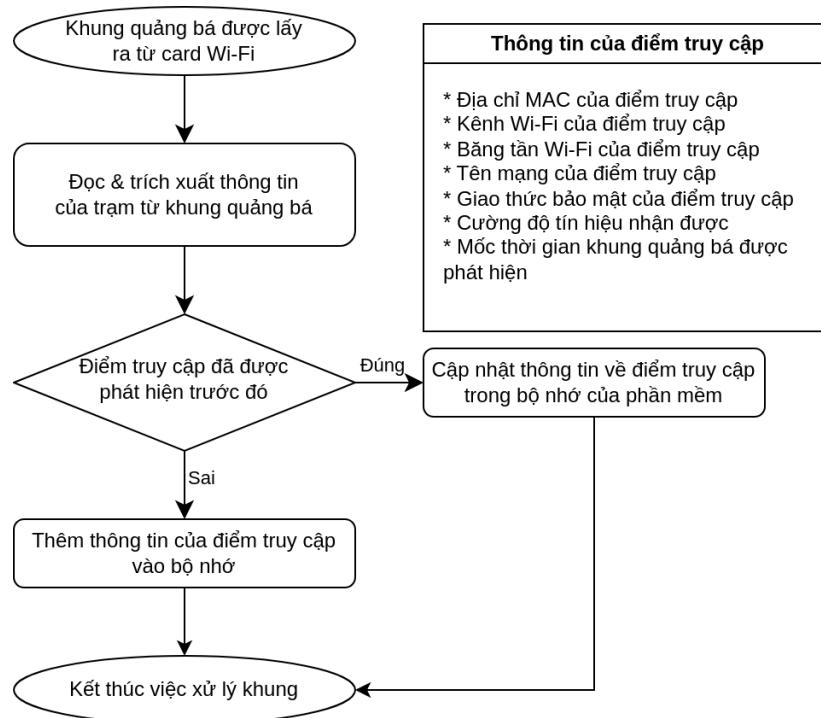
Hình 3.13. Mô hình giao tiếp giữa phần mềm ở không gian người dùng và driver của card Wi-Fi

Khi bắt đầu chạy chương trình sẽ đọc danh sách các card Wi-Fi có sẵn trên thiết bị và lưu vào một cấu trúc dữ liệu gọi là DeviceTable. Khi nhận được yêu cầu, các card phù hợp với mục đích quét sẽ được sao chép vào một bảng con là ScanTable, đại diện cho các card được dùng thực tế trong quá trình thu thập dữ liệu. Với mỗi card Wi-Fi, chương trình mở một phiên làm việc sử dụng thư viện libpcap, kết nối với socket tương ứng để bắt các khung bản tin Wi-Fi.

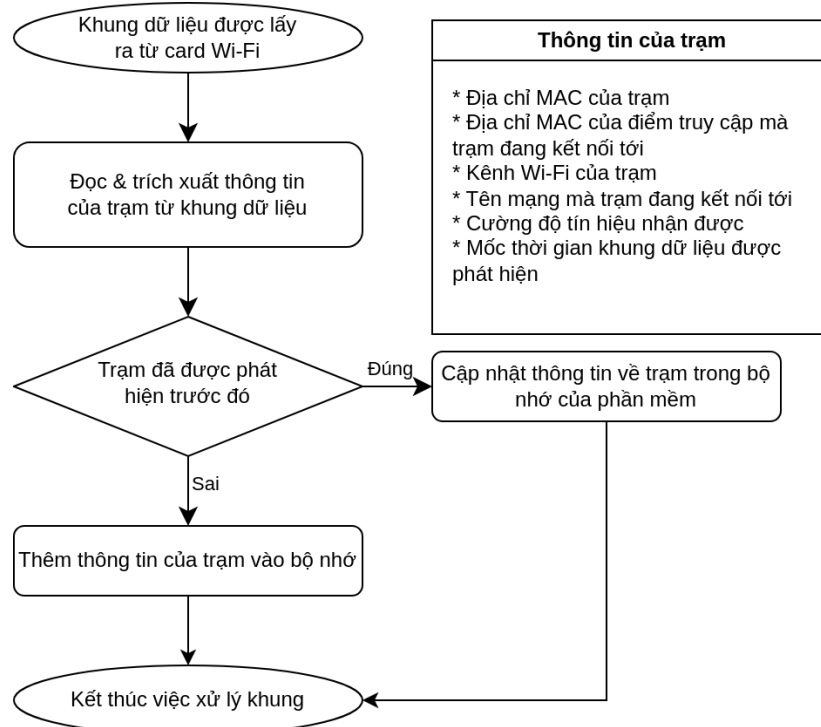
Khi các khung bản tin Wi-Fi đi vào, chúng được xử lý thông qua một hàm chuyên biệt để phân tích nội dung gói tin. Luồng xử lý của hàm này được mô tả ở hình 3.14. và 3.15. Từ đó, chương trình trích xuất các thông tin quan trọng như địa chỉ MAC, SSID, loại thiết bị (AP hoặc STA), cường độ tín hiệu, và các thông số liên quan khác. Dữ liệu được phân tích sẽ được lưu trữ trong một cấu trúc dữ liệu nội bộ, chứa thông tin các thiết bị đã phát hiện.

Cuối cùng, dữ liệu này sẽ được ứng dụng Web cục bộ lấy theo chu kỳ để hiển thị

thông tin cho người dùng qua trình duyệt. Nhờ vậy, người dùng có thể dễ dàng giám sát và kiểm tra các thiết bị đang kết nối hoặc hoạt động gần mạng Wi-Fi của họ một cách trực quan và tiện lợi.



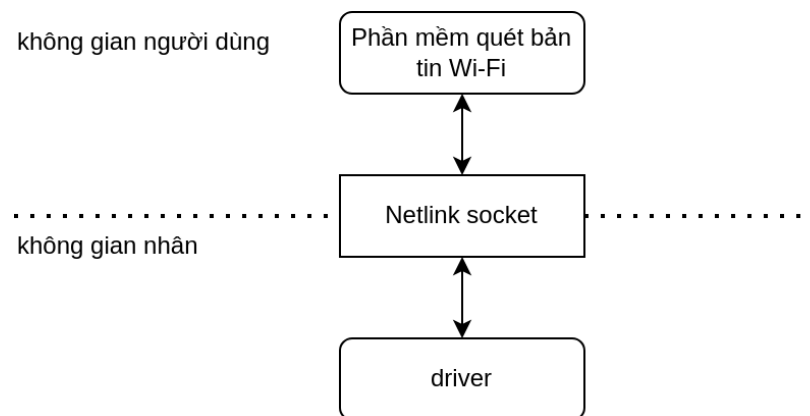
Hình 3.14. Lưu đồ thuật toán cho việc xử lý khung quảng bá



Hình 3.15. Lưu đồ thuật toán cho việc xử lý khung dữ liệu

## b. Điều khiển phần cứng qua Netlink

Phần mềm cần có tính năng lấy thông tin (thông tin về băng tần hỗ trợ, CSI, tên card Wi-Fi,...) và điều khiển tất cả các card Wi-Fi có mặt ở trên thiết bị. Điều này được thực hiện bằng việc phần mềm quét bản tin Wi-Fi, sử dụng thư viện libnl, gửi các tin nhắn chứa yêu cầu cần thực hiện tới Netlink socket. Nhân hệ điều hành khi nhận được tin nhắn sẽ thực hiện các yêu cầu đó và gửi tin nhắn trả lời qua Netlink socket tới phần mềm quét bản tin Wi-Fi. Hình 3.16. mô tả sự tương tác giữa các thành phần để thực hiện chức năng nói trên.



Hình 3.16. Mô hình giao tiếp giữa phần mềm ở không gian người dùng và driver của card Wi-Fi

### c. Thu thập dữ liệu CSI (thành phần thứ hai)

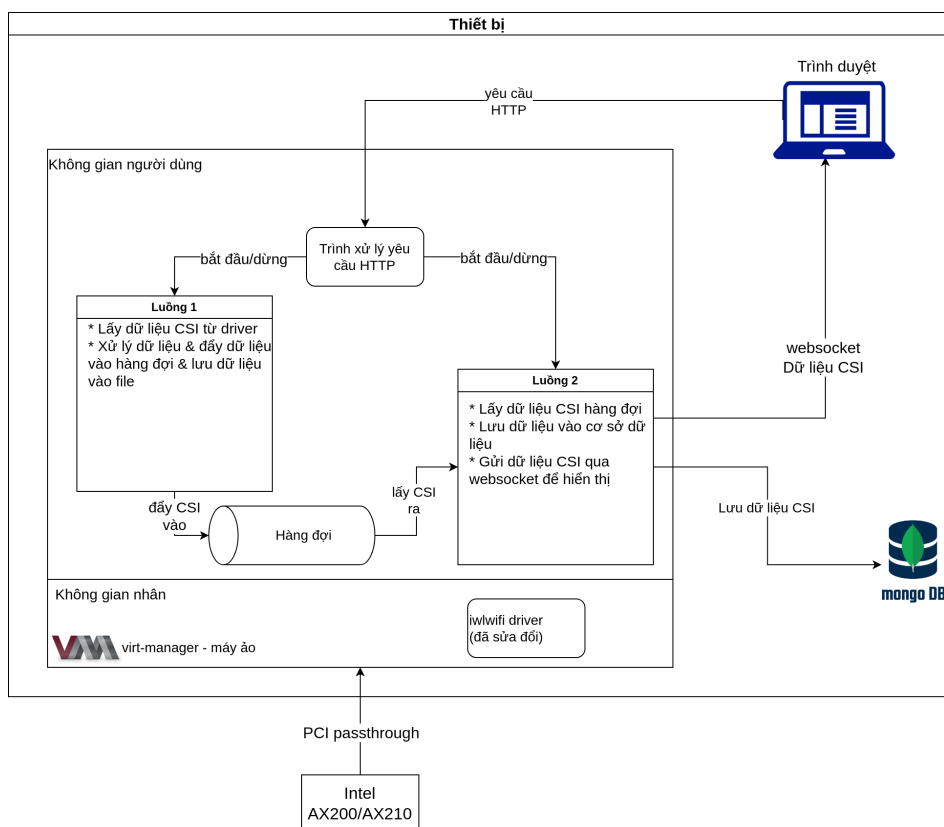
Đồ án có mục tiêu là triển khai tính năng thu thập dữ liệu CSI từ card Wi-Fi Intel AX200/AX210 vì đây là dòng card Wi-Fi phổ biến, có sẵn ở trên nhiều máy tính xách tay hiện nay, giá thành rẻ và hỗ trợ nhiều chuẩn Wi-Fi hỗ trợ nhiều chuẩn Wi-Fi như 802.11a/g/n/ac/ax.

Tính năng này được thực hiện thông qua một hệ thống ảo hóa. Trong kiến trúc này (được mô tả bởi hình 3.17.), card Wi-Fi vật lý được gắn trực tiếp vào máy ảo bằng công nghệ PCI passthrough. Bên trong máy ảo, một phiên bản driver iwlmwifi đã được chỉnh sửa từ FeitCSI cho phép trích xuất dữ liệu CSI từ phần cứng.

Dữ liệu CSI sau khi được lấy từ driver sẽ được xử lý trong không gian người dùng thông qua hai luồng chính. Luồng đầu tiên chịu trách nhiệm thu thập dữ liệu CSI từ driver, xử lý thô (định dạng lại, đánh dấu thời gian), sau đó đẩy dữ liệu vào một hàng đợi nội bộ và đồng thời ghi dữ liệu này vào file để lưu trữ lâu dài hoặc phân tích offline. Luồng thứ hai lấy dữ liệu CSI từ hàng đợi, lưu trữ vào cơ sở dữ liệu MongoDB và truyền tiếp đến giao diện người dùng qua giao thức WebSocket để hiển thị theo thời gian thực.

Người dùng tương tác với hệ thống thông qua một giao diện web. Khi giao diện gửi yêu cầu HTTP đến thành phần này, quá trình thu thập dữ liệu sẽ được khởi động hoặc dừng lại tùy theo yêu cầu. Dữ liệu CSI được gửi qua WebSocket đến trình duyệt

để hiển thị trực quan, phục vụ các mục đích như theo dõi trạng thái kênh không dây, nghiên cứu hành vi tín hiệu hoặc phục vụ các ứng dụng định vị và học máy.



Hình 3.17. Sơ đồ kiến trúc hệ thống thu thập và hiển thị dữ liệu CSI

#### d. Cung cấp API cho các mô-đun khác

Phần mềm quét bản tin Wi-Fi cần có khả năng phối hợp và giao tiếp với các ứng dụng khác trong cùng hệ thống. Trong đồ án này, việc giao tiếp được thực hiện thông qua RESTful API, được xây dựng bằng thư viện Crow – một HTTP framework cho ngôn ngữ C++.

Trong kiến trúc của cả hai thành phần của Phần mềm quét bản tin Wi-Fi, bộ định tuyến yêu cầu HTTP hay trình xử lý yêu cầu HTTP được thực hiện bằng thư viện Crow. Crow hoạt động như một cầu nối giữa giao diện web và phần lõi xử lý gói tin Wi-Fi trong chương trình, cho phép người dùng tương tác với hệ thống một cách thuận tiện và thời gian thực thông qua giao diện trình duyệt.

#### 3.2.3.2 Phần mềm quét bản tin Wi-Fi

#### 3.2.3.3 Phương thức kết nối

#### 3.2.3.4 Sequence Diagram

### Kết luận chương

Qua quá trình thiết kế chi tiết, hệ thống/sản phẩm đã được xây dựng dựa trên các yêu cầu kỹ thuật và chức năng đã phân tích. Mô hình tổng thể, thiết kế phần cứng và phần

mềm cũng như phương thức tích hợp đảm bảo sự vận hành đồng bộ và hiệu quả của hệ thống. Thiết kế này tạo nền tảng vững chắc cho giai đoạn triển khai thực tế và phát triển tiếp theo của dự án.

## CHƯƠNG 4. TRIỂN KHAI VÀ KIỂM THỬ

### Mở đầu chương

Chương 4 chúng em trình bày quá trình xây dựng thực tế và thử nghiệm hệ thống/sản phẩm. Nội dung bao gồm quy trình triển khai, các hoạt động thử nghiệm và đo kiểm, đánh giá kết quả thực tế so với thiết kế ban đầu, cũng như các vấn đề phát sinh và cách khắc phục trong quá trình thực hiện.

### 4.3 Phần mềm quản lý

#### 4.3.1 Quy trình triển khai

##### 4.3.1.1 Cấu hình và triển khai ThingsBoard

ThingsBoard được cài đặt và cấu hình trên máy chủ local, sử dụng cổng mặc định 8080 để truy cập giao diện quản trị. Quá trình cấu hình bao gồm các bước chính sau:

- Cài đặt ThingsBoard: Cài đặt Java 17 (OpenJDK) làm môi trường chạy, sau đó cài đặt ThingsBoard Community Edition bằng gói cài đặt phù hợp với hệ điều hành (Ubuntu Server). Sau khi cài đặt, ThingsBoard được cấu hình để sử dụng cơ sở dữ liệu PostgreSQL, luồng dữ liệu đẩy vào queue của Thingsboard sử dụng In-memory Queue đảm bảo lưu trữ dữ liệu thiết bị ổn định [15].

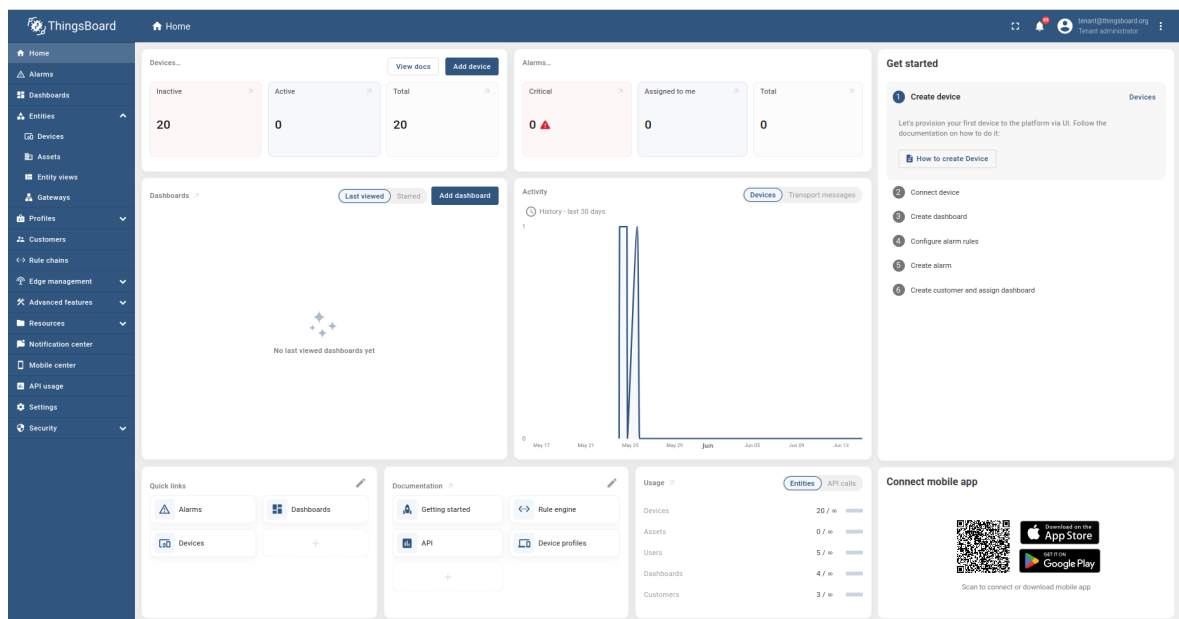
```
OpenJDK 64-Bit Server VM warning: Option UseBiasedLocking was deprecated in version 15.0 and will likely be removed in a future release.

=====
:: ThingsBoard ::      (v4.0.1)
=====

Starting ThingsBoard Installation...
Installing DataBase schema for entities...
Installing SQL DataBase schema part: schema-entities.sql
Installing SQL DataBase schema indexes part: schema-entities-idx.sql
Installing SQL DataBase schema PostgreSQL specific indexes part: schema-entities-idx-psql-addon.sql
Installing SQL DataBase schema views and functions: schema-views-and-functions.sql
Successfully executed query: DROP VIEW IF EXISTS device_info_view CASCADE;
Successfully executed query: CREATE OR REPLACE VIEW device_info_view AS SELECT * FROM device_info_active_attribute_view;
Installing DataBase schema for timeseries...
Installing SQL DataBase schema part: schema-ts-psql.sql
Successfully executed query: CREATE TABLE IF NOT EXISTS ts_kv_indefinite PARTITION OF ts_kv DEFAULT;
```

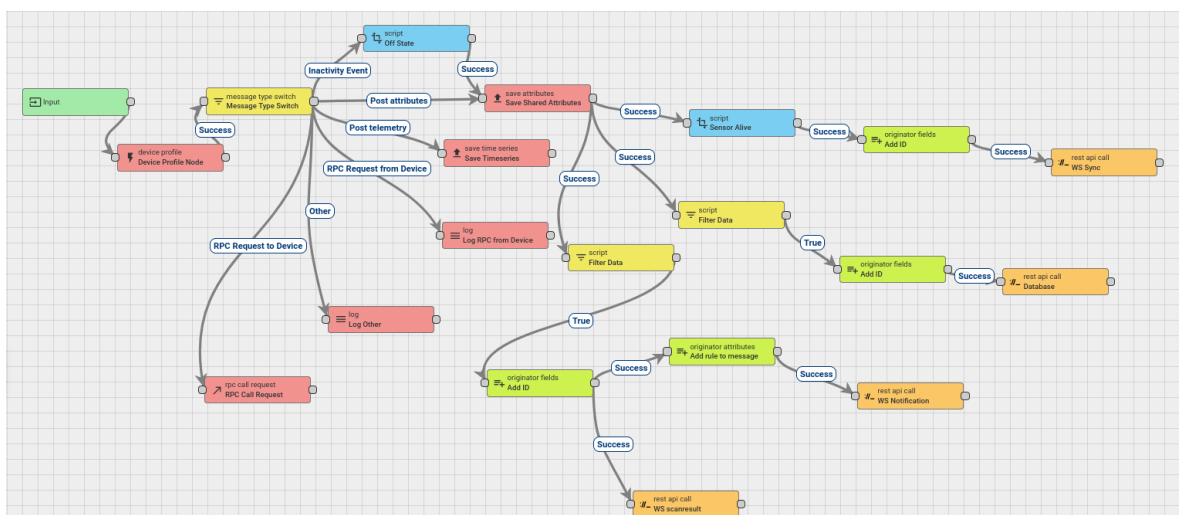
Hình 4.1. Cài đặt và cấu hình thành công Thingsboard

- Khởi động dịch vụ và truy cập: Sau khi hoàn tất cài đặt và cấu hình cơ sở dữ liệu, dịch vụ ThingsBoard được khởi động. Người dùng có thể truy cập giao diện quản trị tại địa chỉ <http://localhost:8080/> trên trình duyệt web, đăng nhập bằng tài khoản tenant.



Hình 4.2. Giao diện Thingsboard được chạy trên localhost port 8080

## Cấu hình Rule Chain cho Thingsboard



Hình 4.3. Cấu trúc Root Rule Chain sử dụng

Luồng dữ liệu từ thiết bị đến hệ thống ThingsBoard được xử lý như sau:

- Nguồn dữ liệu: Dữ liệu được gửi từ thiết bị thông qua LocalBE bao gồm 3 trường chính:
  - scanresult – Kết quả quét từ thiết bị, định dạng JSON.
  - state – Trạng thái của thiết bị (0: idle, 1: active, 2: unknown).
  - sync – Chế độ đồng bộ: false (real-time), true (lịch sử cần đồng bộ).



scanresult	sync	state	Hành động
Có	false	-	Gửi dữ liệu đến WebSocket (Scanresult, Notification)
Có	true	-	Gửi dữ liệu đến REST API để lưu lịch sử vào DB
Không	-	Có	Cập nhật trạng thái thiết bị (attributes), gửi WS Sync

*Bảng 4.1. Quy tắc xử lý dữ liệu trong Rule Chain*

- Đầu ra dữ liệu: Dữ liệu sau khi được xử lý ở Rule Chain sẽ được chuyển tiếp tới Backend thông qua các REST API Call Node để xử lý tiếp, hoặc sẽ nhận các RPC Call từ Backend để yêu cầu xuống Device bên dưới để xử lý.

#### 4.3.1.2 Triển khai Backend API

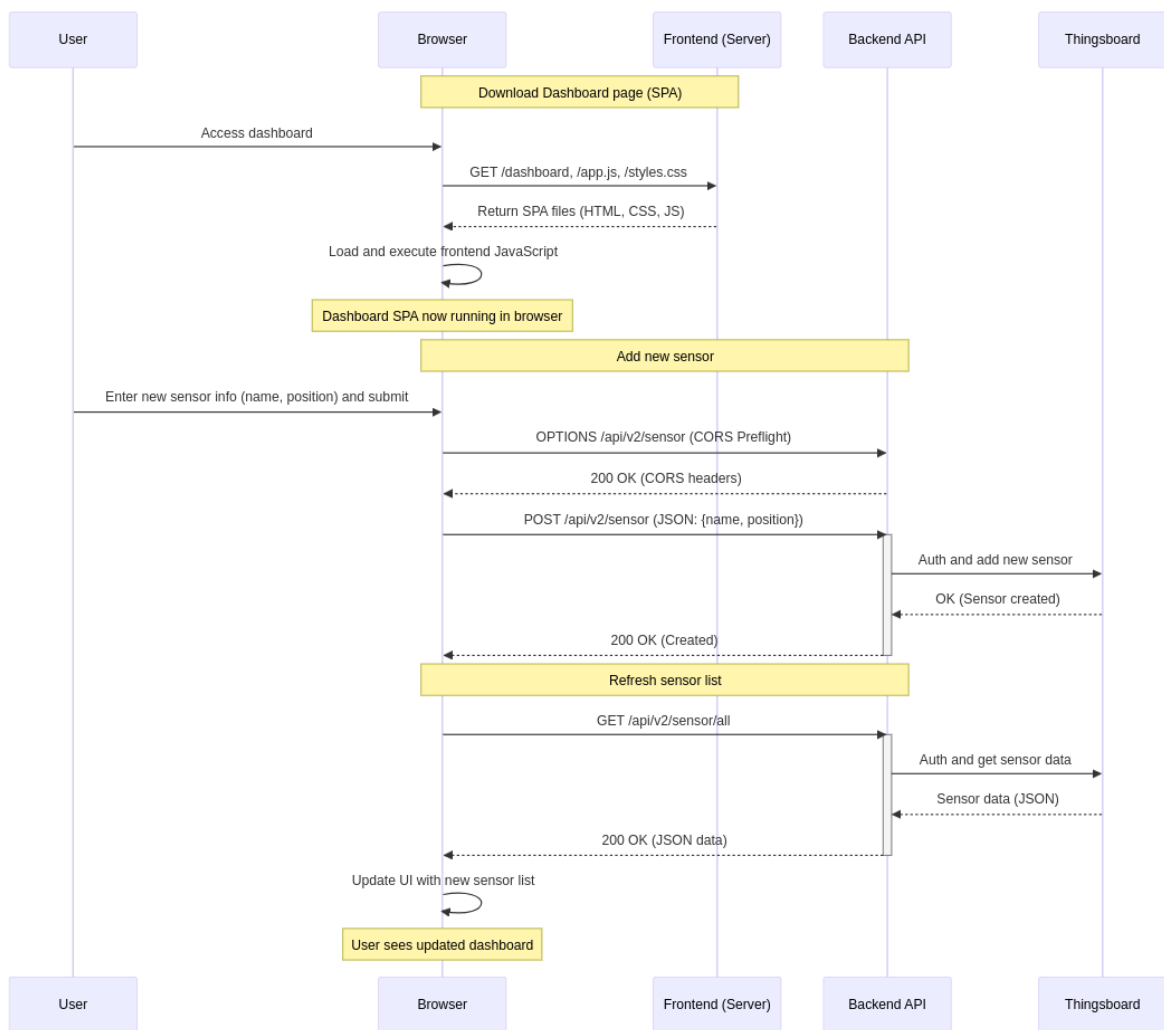
Xây dựng API trung gian để nhận, xử lý dữ liệu từ ThingsBoard, lưu trữ vào cơ sở dữ liệu (PostgreSQL) và cung cấp các endpoint phục vụ truy vấn, quản lý thiết bị. Backend được xây dựng dựa trên FastAPI framework được hỗ trợ bởi ngôn ngữ lập trình python chạy tại địa chỉ <http://localhost:4000/>. Trong Backend gồm có rất nhiều API xử lý và chuyển tiếp dữ liệu, được liệt kê dưới bảng sau:

Phương thức	Endpoint	Chức năng	Nhóm
GET	/sensor/all	Lấy danh sách tất cả các sensor	Sensor
POST	/sensor	Thêm sensor mới	Sensor
PUT	/sensor/{id}	Cập nhật thông tin sensor theo ID	Sensor
DELETE	/sensor/{id}	Xóa sensor theo ID	Sensor
GET	/sensor/{id}/cards	Lấy danh sách card mạng của sensor	Cards
PUT	/sensor/{id}/cards	Cập nhật card của sensor	Cards
POST	/sensor/{id}/start	Bắt đầu scan	Scan Control
POST	/sensor/{id}/stop	Dừng scan	Scan Control
POST	/db/aps	Truy vấn danh sách AP	Query DB
POST	/db/clients	Truy vấn danh sách STA	Query DB

POST	/db/ap/clients	Truy vấn STA của 1 AP tại 1 thời điểm	Query DB
POST	/db/notifications	Truy vấn danh sách Notification	Query DB
POST	/db/history/aps	Truy vấn lịch sử các AP	Query DB
POST	/db/history/clients	Truy vấn lịch sử các STA	Query DB
POST	/ws/[id]/scanresult	Gửi kết quả quét thời gian thực	Websocket
POST	/ws/sync	Đồng bộ thông tin sensor thời gian thực	Websocket
POST	/ws/noti	Gửi thông báo từ thời gian thực	Websocket
PUT	/sensor/{id}/db/info	Cập nhật metadata của sensor trong DB	Update DB
POST	/db/add	Lưu trữ data từ sensor vào DB	Update DB
POST	/api/auth/login	Đăng nhập vào giao diện người dùng	User
PUT	/user/edit	Cập nhật thông tin người dùng	User

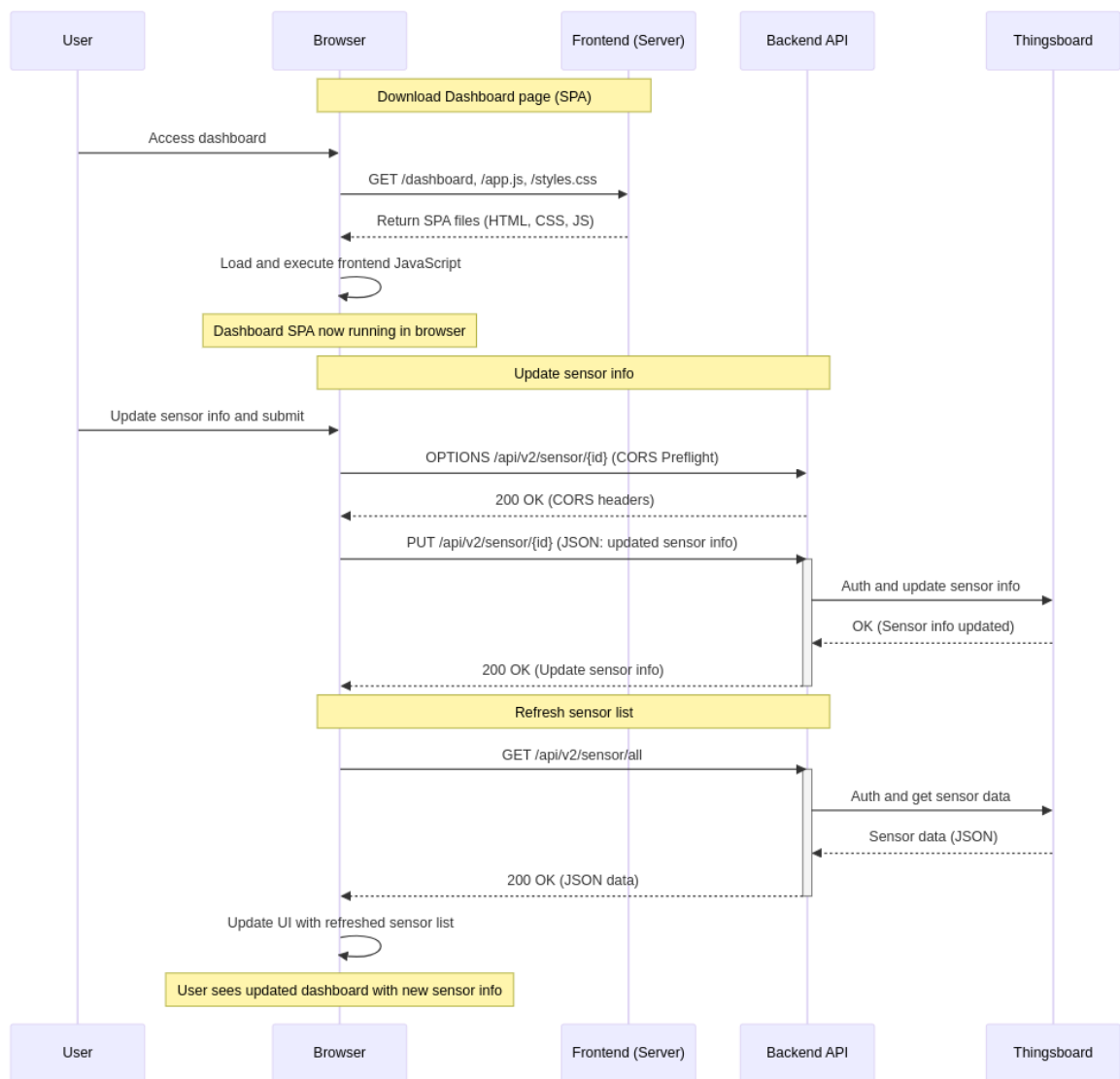
Dưới đây là các sequence diagram giữa các module trong phần mềm quản lý, được phân loại theo nhóm chức năng:

- Nhóm Sensor
  - Luồng thêm sensor mới (Lấy danh sách tất cả các sensor)



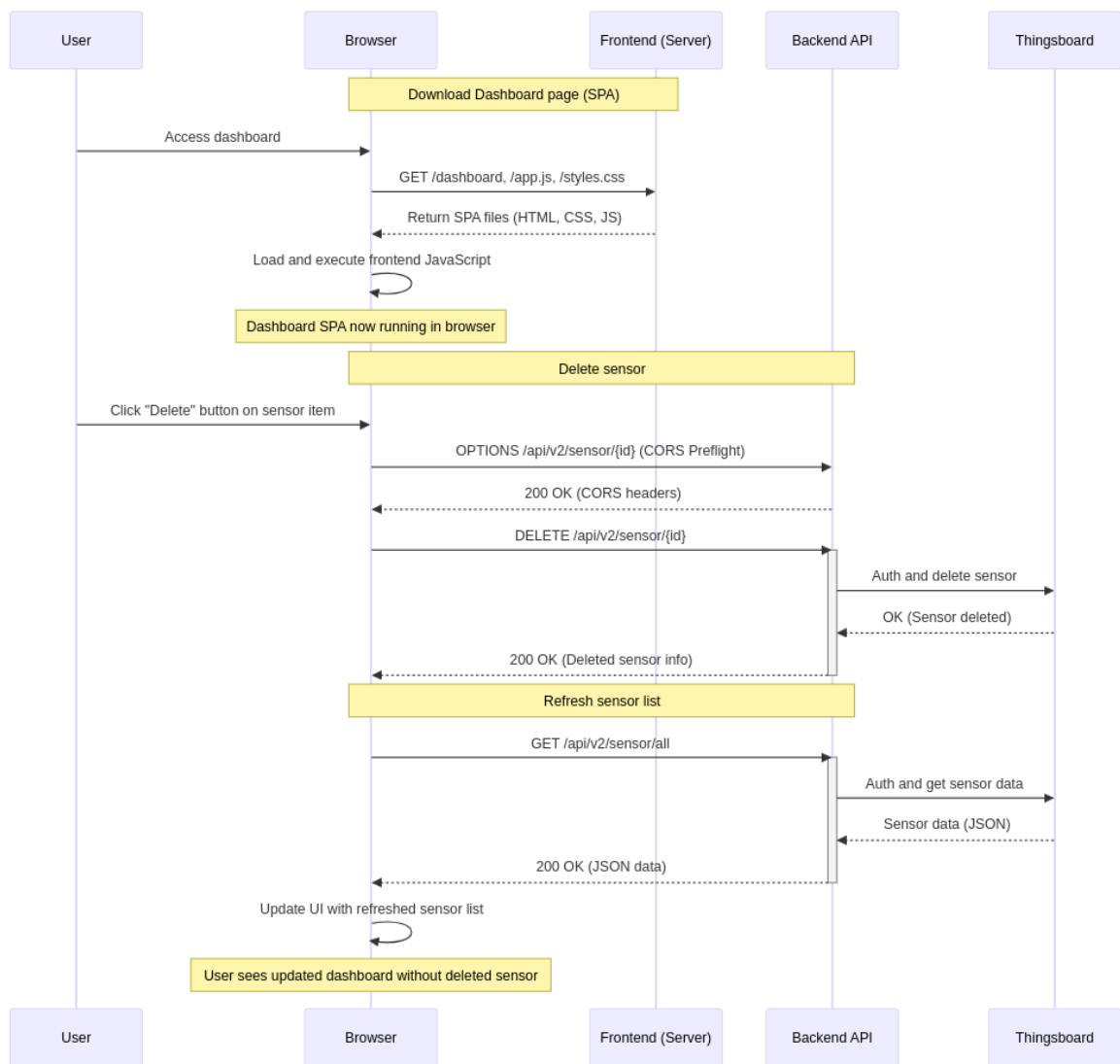
*Hình 4.4. Luồng thêm sensor*

– Luồng cập nhật thông tin sensor theo ID (Lấy danh sách tất cả các sensor)



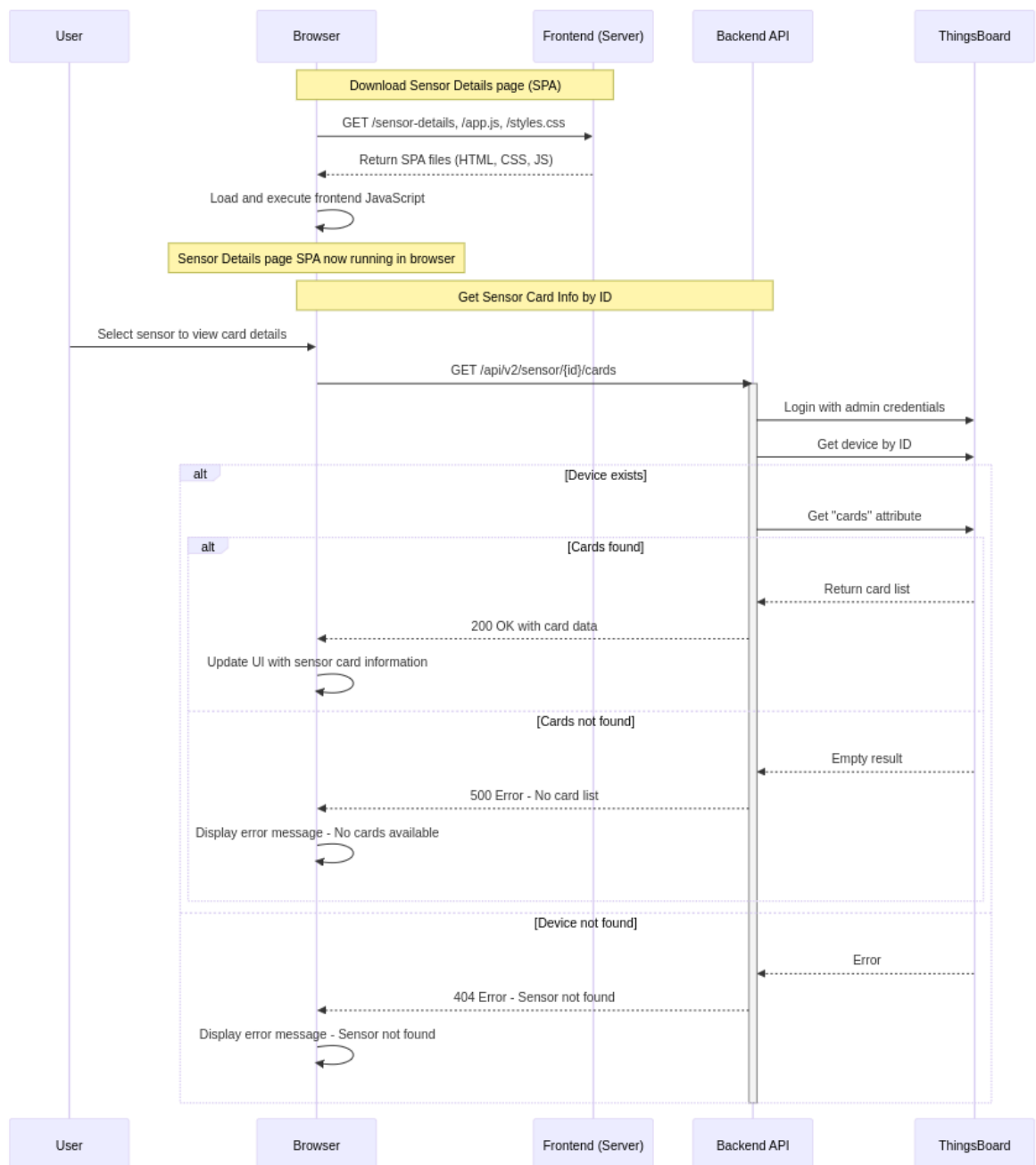
Hình 4.5. Luồng cập nhật sensor

- Luồng xóa sensor theo ID (Lấy danh sách tất cả các sensor)



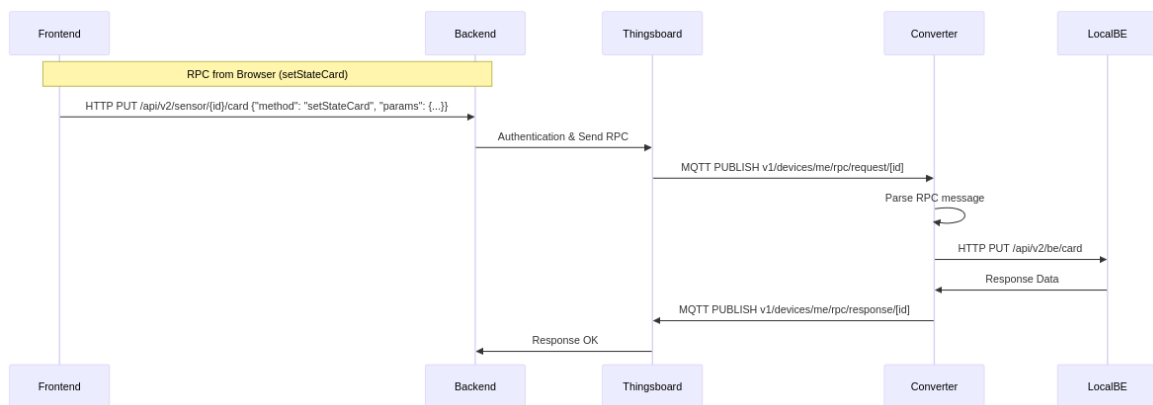
Hình 4.6. Luồng xóa sensor

- Nhóm Cards
  - Luồng lấy danh sách card mạng của sensor



Hình 4.7. Luồng lấy danh sách card mạng của sensor

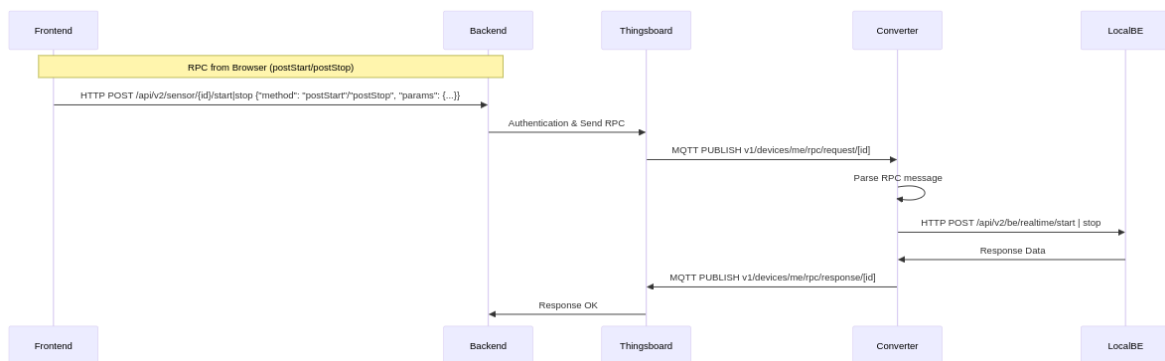
– Luồng cập nhật card của sensor



Hình 4.8. Luồng thiết lập trạng thái card từ người dùng

- Nhóm Scan Control

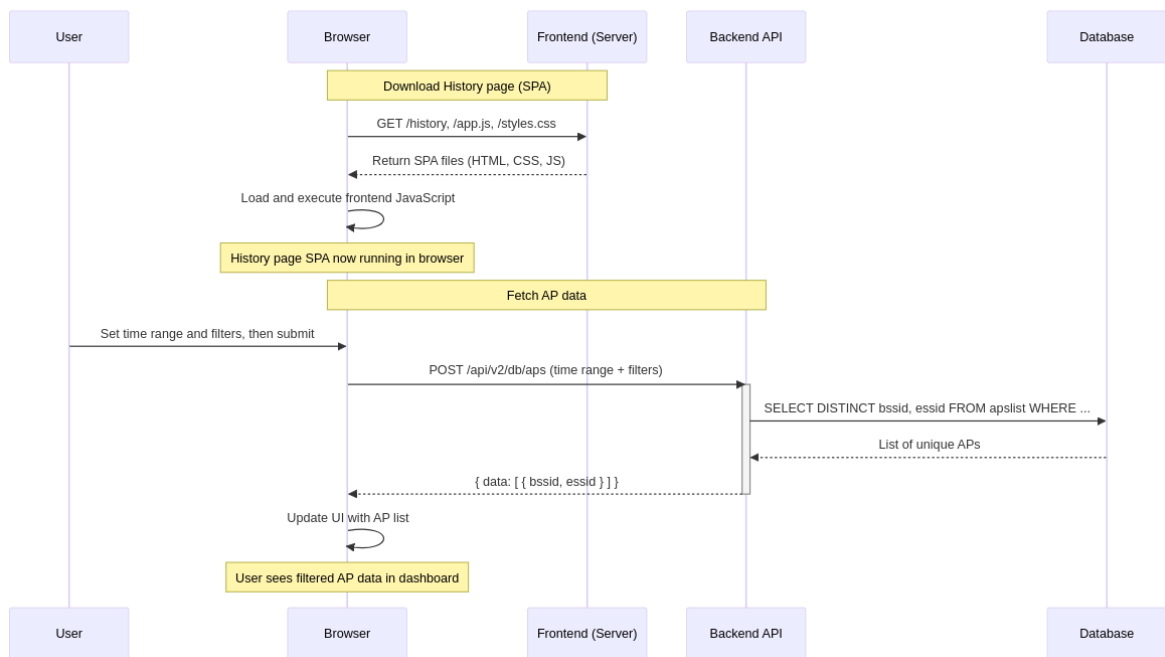
- Luồng bắt đầu hoặc dừng scan



Hình 4.9. Luồng bật tắt nhận hiển thị kết quả quét từ người dùng

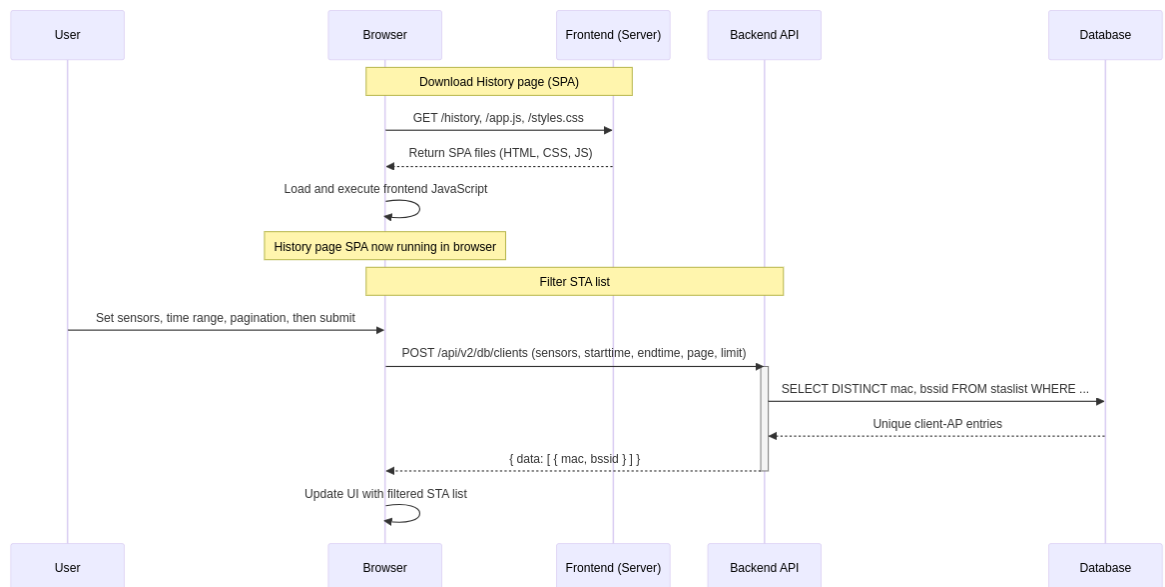
- Nhóm Query DB

- Luồng truy vấn danh sách AP



Hình 4.10. Luồng truy vấn danh sách AP

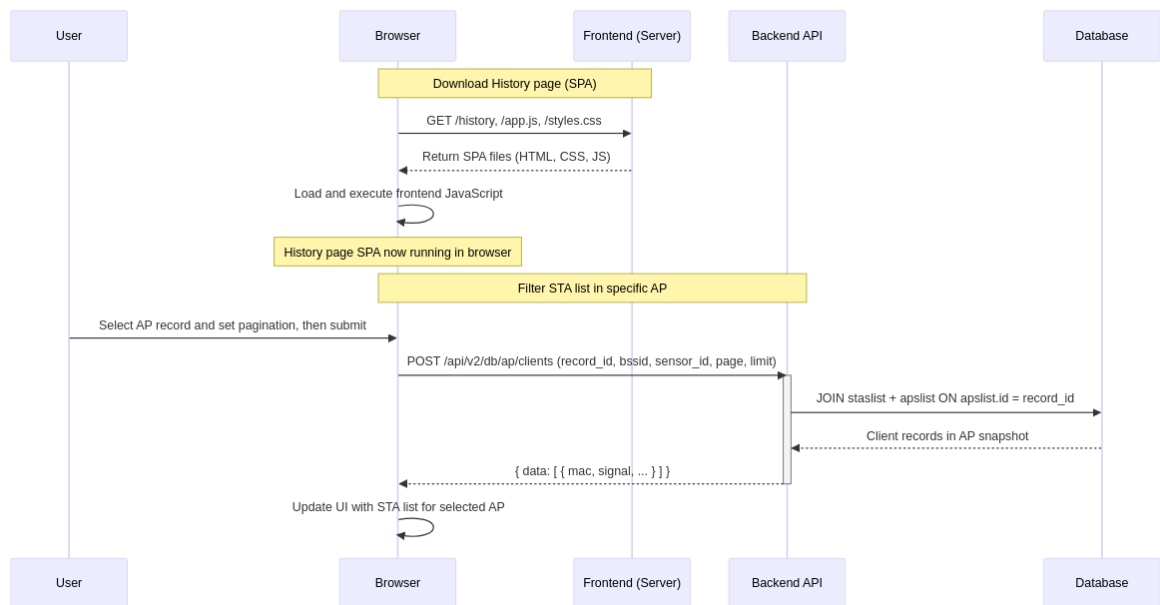
#### – Luồng truy vấn danh sách STA



Hình 4.11. Luồng truy vấn danh sách STA

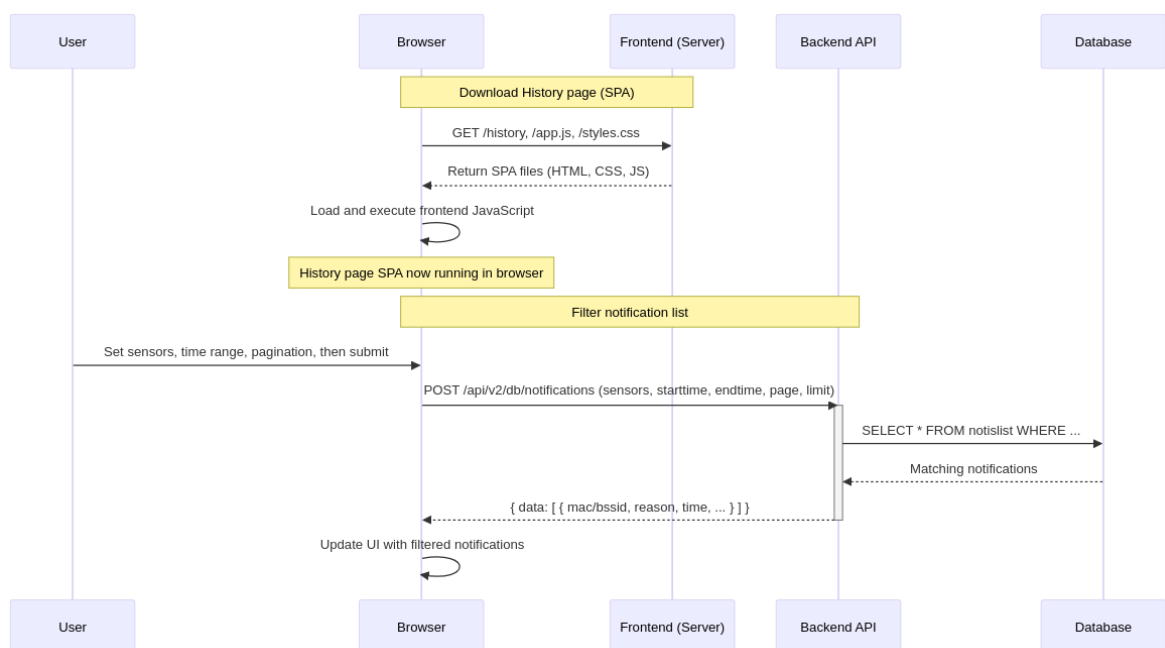
#### – Luồng truy vấn STA của 1 AP tại 1 thời điểm





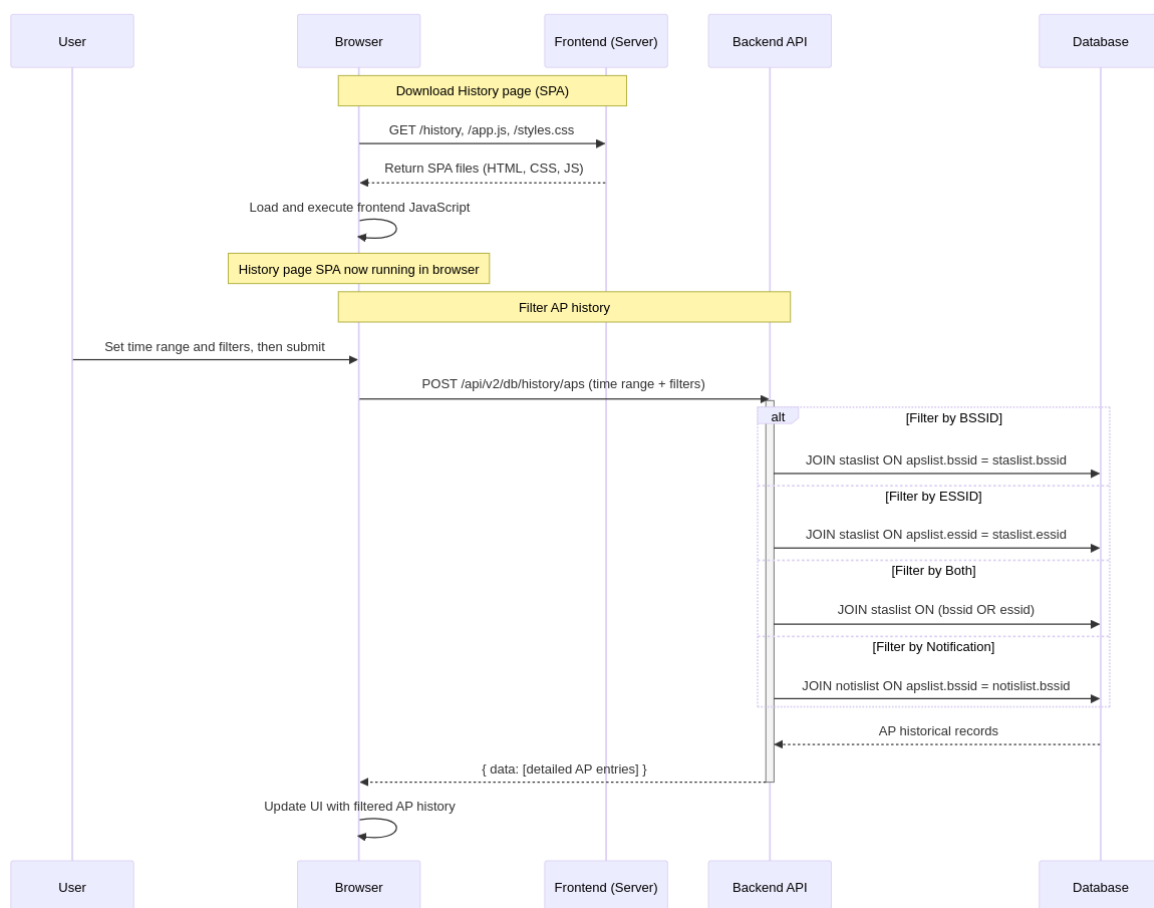
Hình 4.12. Luồng truy vấn STA của 1 AP tại 1 thời điểm

#### – Luồng truy vấn danh sách Notification



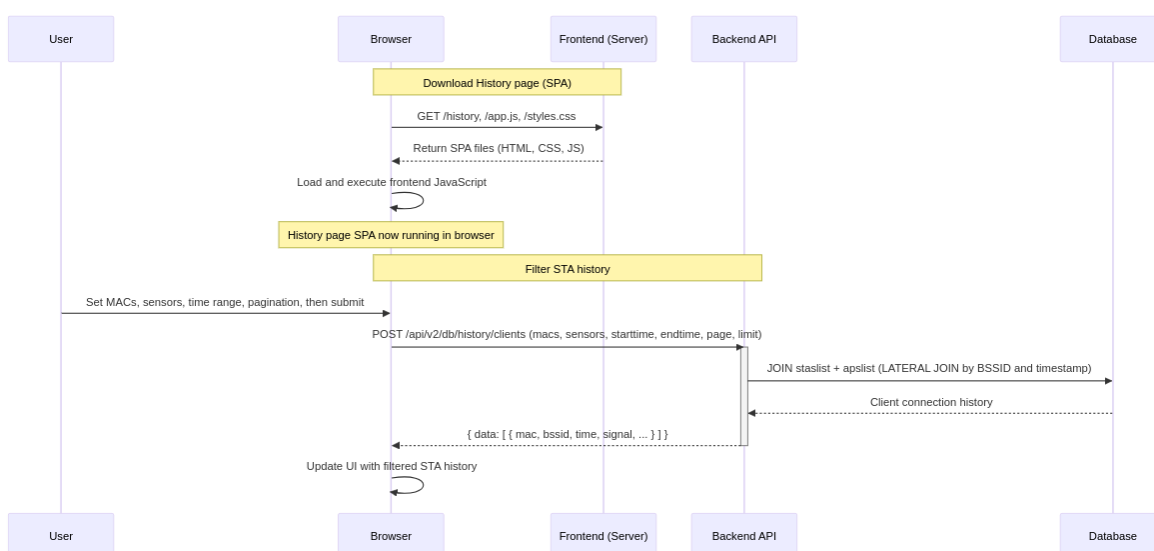
Hình 4.13. Luồng truy vấn danh sách Notification

#### – Luồng truy vấn lịch sử AP



Hình 4.14. Luồng truy vấn lịch sử AP

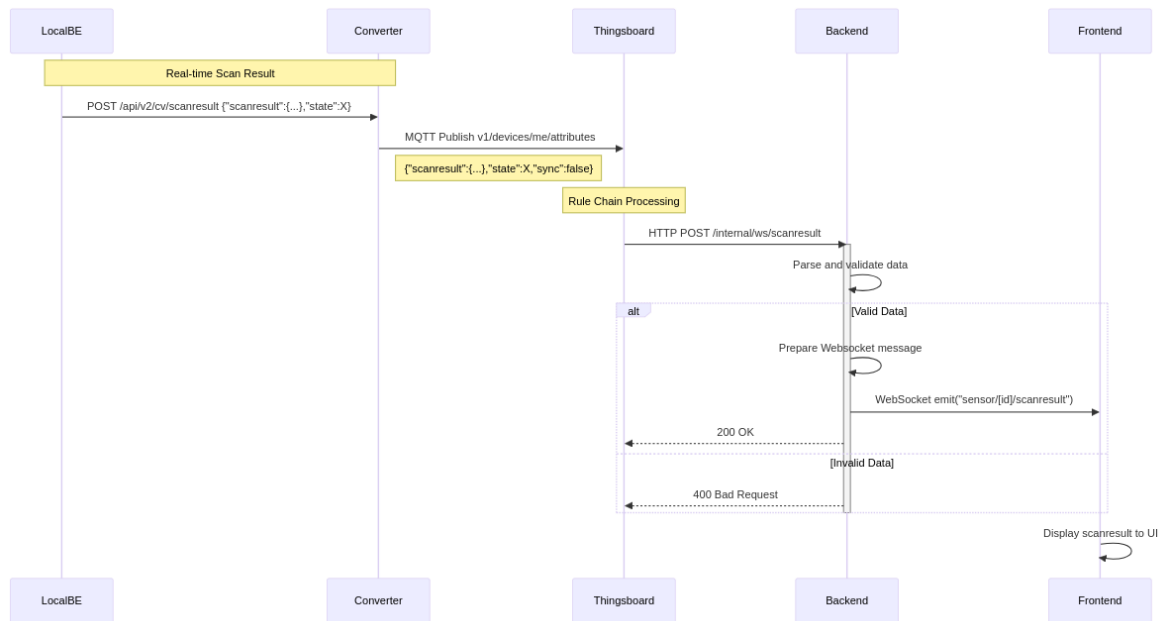
## – Luồng truy vấn lịch sử STA



Hình 4.15. Luồng truy vấn lịch sử STA

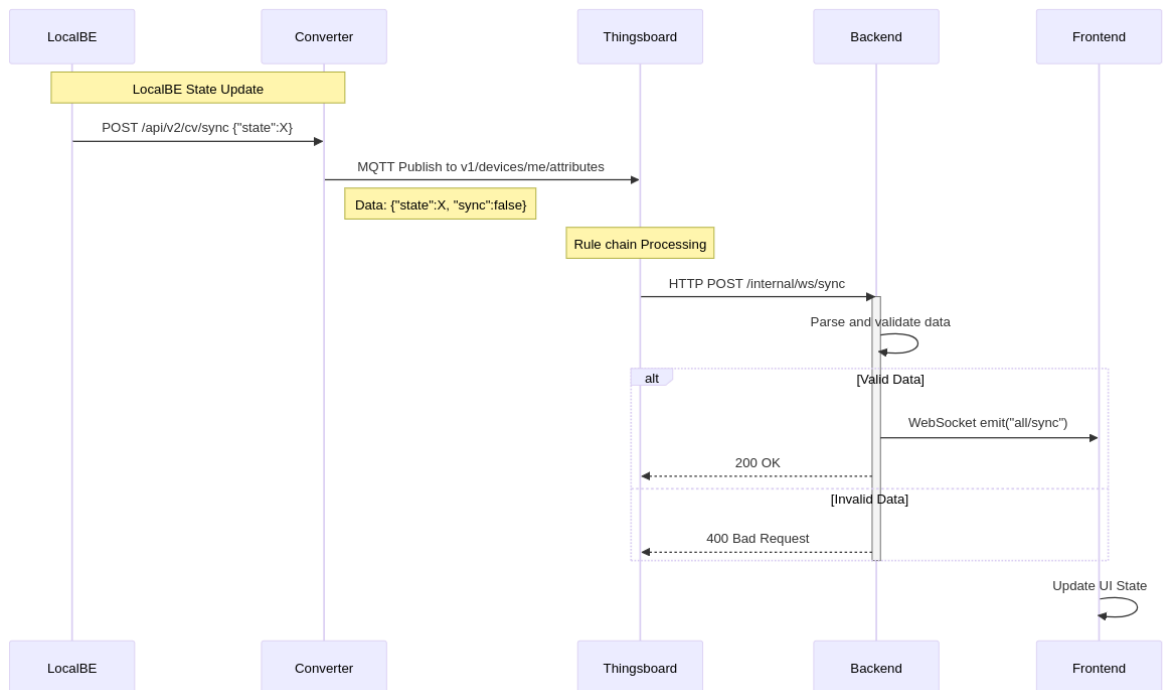
## • Nhóm Websocket

### – Luồng gửi kết quả quét thời gian thực



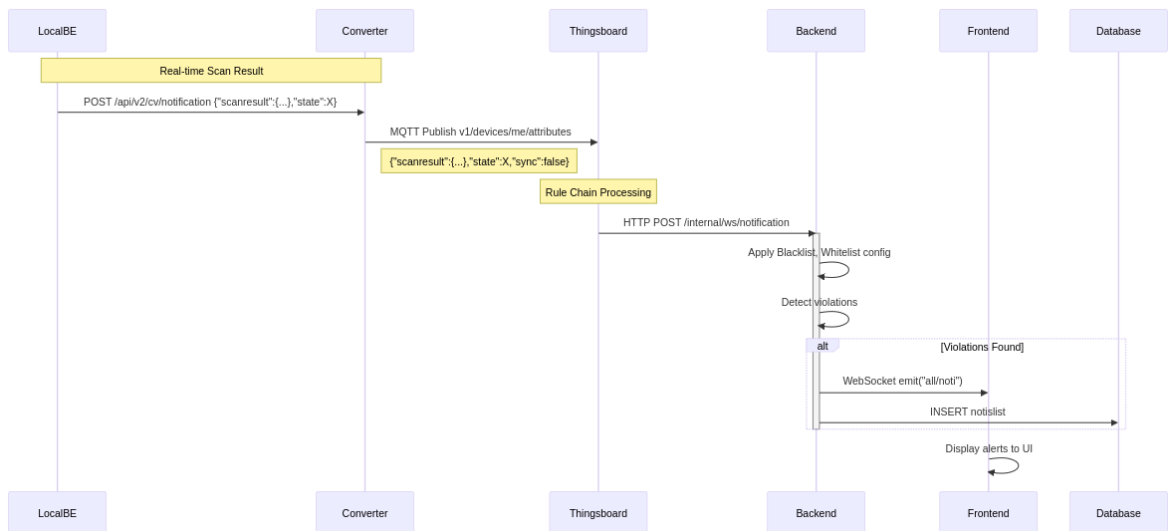
Hình 4.16. Luồng cập nhật kết quả quét realtime trên giao diện hiển thị

#### – Luồng đồng bộ thông tin sensor thời gian thực



Hình 4.17. Luồng cập nhật trạng thái thiết bị realtime trên giao diện hiển thị

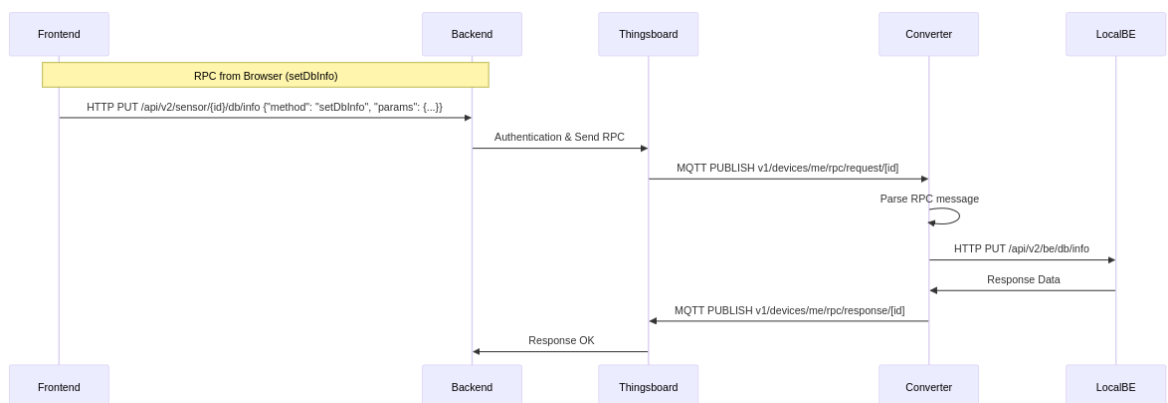
#### – Luồng gửi thông báo từ thời gian thực



Hình 4.18. Luồng cập nhật thông báo realtime trên giao diện hiển thị

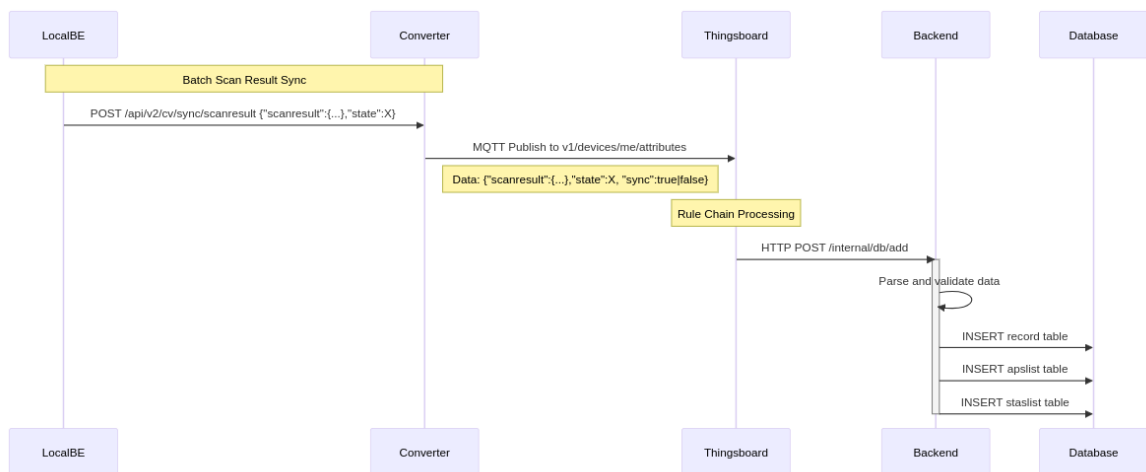
- Nhóm Update DB

- Luồng cập nhật metadata của sensor trong DB



Hình 4.19. Luồng cập nhật thông tin thiết bị vào database ngoại từ người dùng

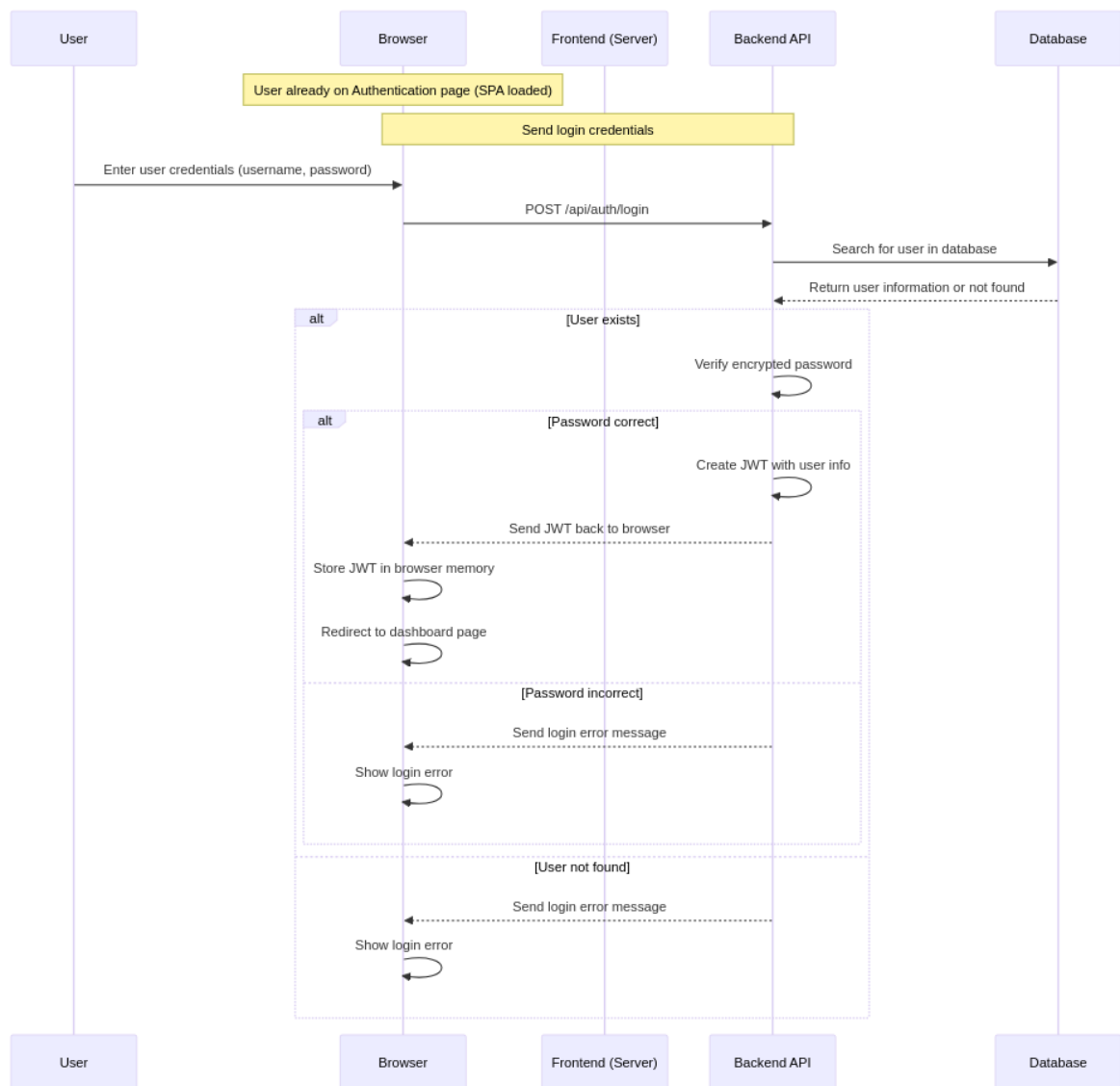
- Luồng lưu trữ data từ sensor vào DB



*Hình 4.20. Luồng lưu kết quả quét vào database ngoại*

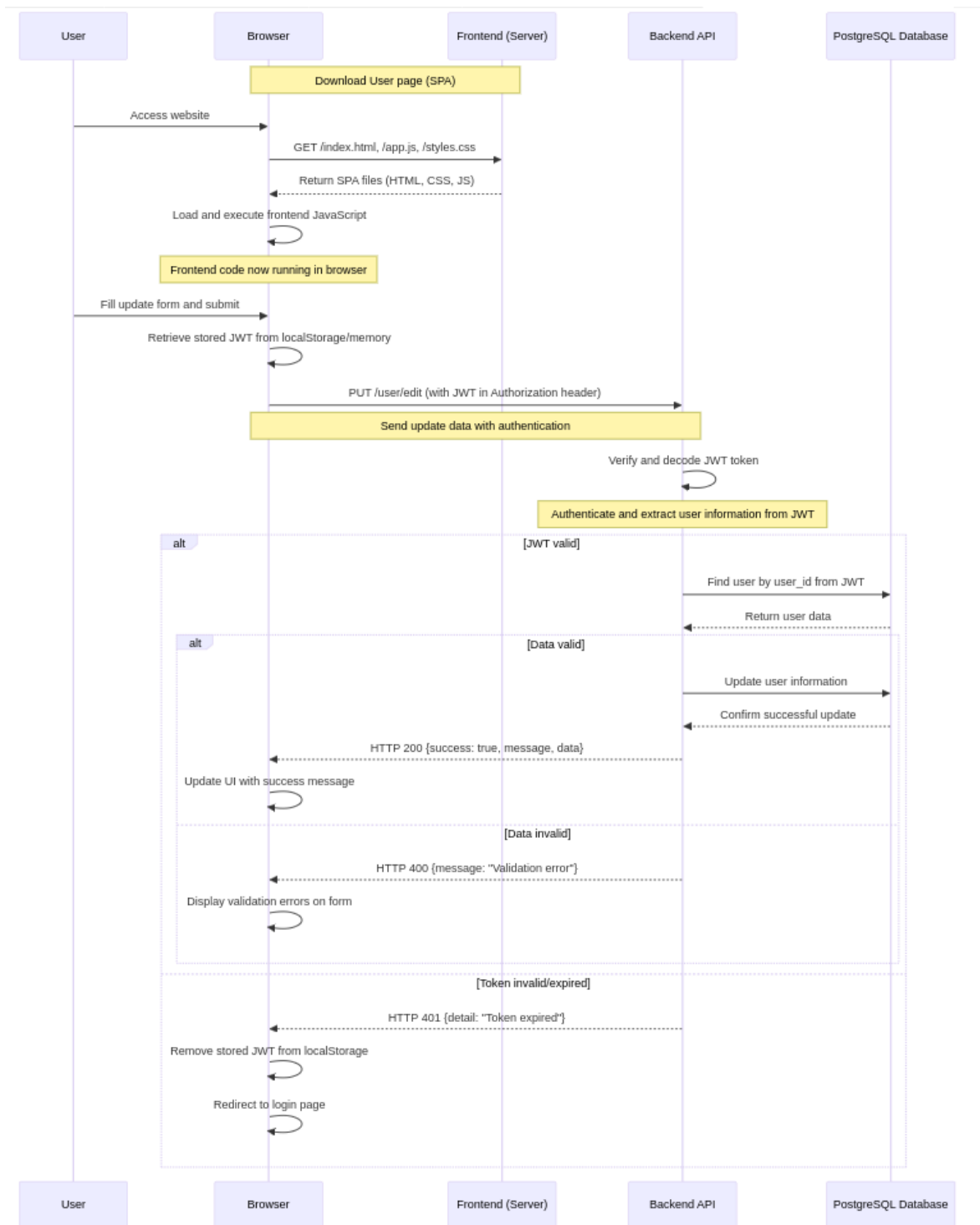
- Nhóm Users

- Luồng đăng nhập giao diện người dùng



Hình 4.21. Luồng đăng nhập giao diện người dùng

– Luồng cập nhật thông tin người dùng



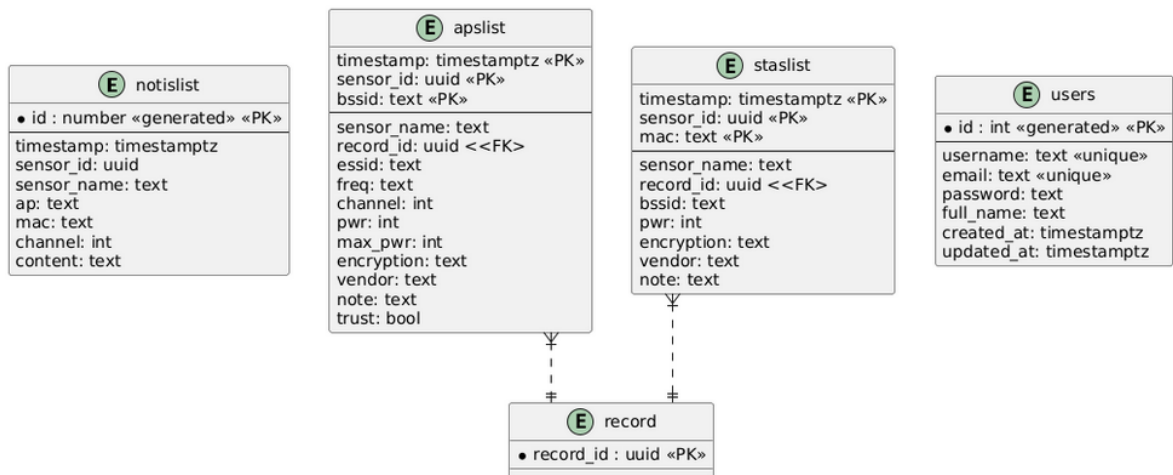
Hình 4.22. Luồng cập nhật thông tin người dùng

#### 4.3.1.3 Kết nối và cấu hình Database

Thiết lập cơ sở dữ liệu để lưu trữ thông tin thiết bị, lịch sử quét, các sự kiện và cảnh báo. Đảm bảo backup. Database xây dựng dựa trên PostgreSQL và chạy tại địa chỉ <http://localhost:5432/>

```
# DB Configuration
export DATABASE_TS_TYPE=sql
export SPRING_DATASOURCE_URL=jdbc:postgresql://localhost:5432
export SPRING_DATASOURCE_USERNAME=postgres
export SPRING_DATASOURCE_PASSWORD=123
```

Hình 4.23. Cấu hình PostgreSQL



Hình 4.24. Các bảng trong cơ sở dữ liệu ngoài

#### 4.3.1.4 Triển khai giao diện người dùng

Xây dựng giao diện web cho phép người dùng theo dõi trạng thái thiết bị, dữ liệu quét wifi theo thời gian thực, thực hiện các thao tác quản lý từ xa. Frontend xây dựng dựa trên ngôn ngữ lập trình Next.js và chạy tại địa chỉ <http://localhost:3000/>

- Thiết kế Trang Login
- Thiết kế Trang Dashboard
- Thiết kế Trang Sensor Details
- Thiết kế Trang History
- Thiết kế Trang User

#### 4.3.2 Thử nghiệm và đo kiểm

Dưới đây là một số kịch bản thử nghiệm và đo kiểm của em trên Hệ thống quản lý:

- Kiểm tra khả năng lưu trữ dữ liệu khi mất kết nối: Khi Thiết bị dò quét wifi mất kết nối tới server (do mạng không ổn định hoặc do server chưa được bật), dữ liệu

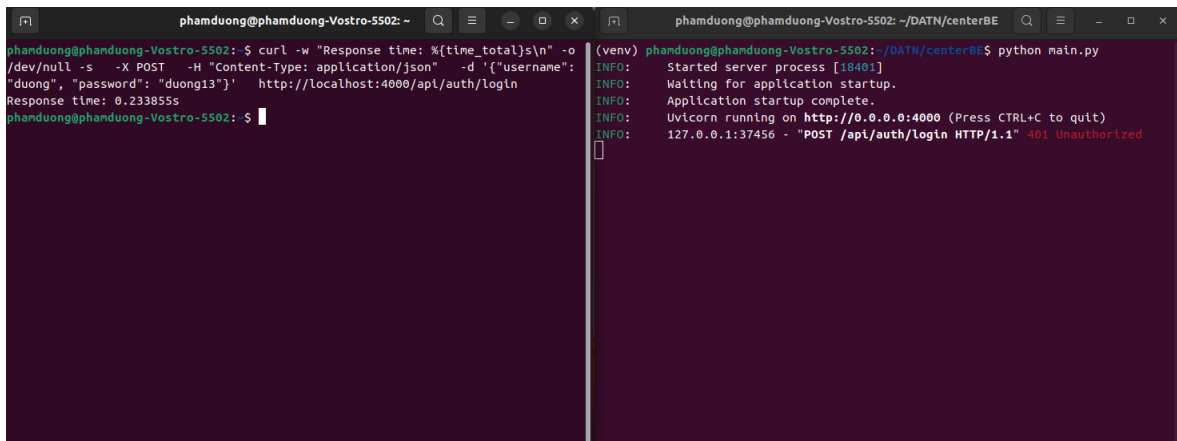


quét được ở dưới thiết bị sẽ được lưu trữ tại cơ sở dữ liệu khác dưới Local (tối đa 1GB). khi thiết bị kết nối lại được với server thì dữ liệu đó sẽ được đẩy dần vào cơ sở dữ liệu ở trên Server.

Cách kiểm tra:

- Test mất kết nối: Ngắt kết nối mạng hoặc tắt server trong quá trình quét
  - Kiểm tra storage local: Xác minh dữ liệu được lưu trong database local
  - Test đồng bộ: Khi kết nối lại, kiểm tra dữ liệu được đẩy lên server theo đúng thứ tự thời gian
- Đo thời gian response từ các module (Device, Thingsboard, Backend, Database, Frontend), ở đây em sử dụng 1 công cụ dòng lệnh cURL (client URL) được hỗ trợ sẵn trên Linux OS để đo thời gian phản hồi của các API (ở Backend). Ở đây sử dụng 2 Terminal: Terminal bên trái để đo tính tổng thời gian từ lúc HTTP request cho tới khi nhận được HTTP response tương ứng (giữa HTTP request và HTTP response sẽ được Backend xử lý thông qua các tương tác tới Thingsboard, Database, Frontend tùy thuộc vào loại API tương ứng), Terminal bên phải để chạy chương trình FastAPI của Backend để kiểm tra xem HTTP request có hợp lệ hay không. Dưới đây là ví dụ đo thời gian response khi xác thực người dùng (Frontend – Backend – Database):

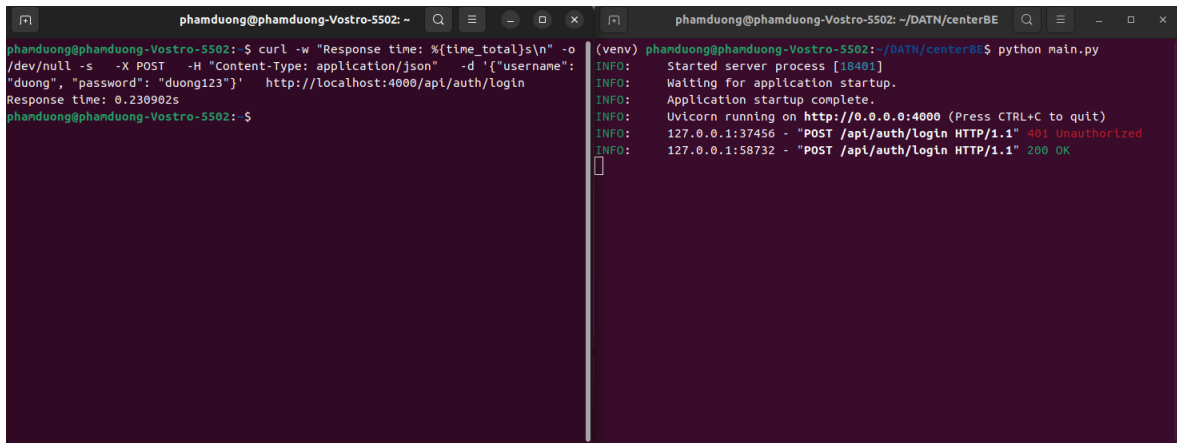
Đo thời gian phản hồi API xác thực người dùng thất bại:



```
phamduong@phamduong-Vostro-5502: ~  
phamduong@phamduong-Vostro-5502:~$ curl -w "Response time: %{time_total}s\n" -o /dev/null -s -X POST -H "Content-Type: application/json" -d '{"username": "duong", "password": "duong13"}' http://localhost:4000/api/auth/login  
Response time: 0.233855s  
phamduong@phamduong-Vostro-5502:~$  
  
(venv) phamduong@phamduong-Vostro-5502:~/DATN/centerBE$ python main.py  
INFO: Started server process [18401]  
INFO: Waiting for application startup.  
INFO: Application startup complete.  
INFO: Uvicorn running on http://0.0.0.0:4000 (Press CTRL+C to quit)  
INFO: 127.0.0.1:37456 - "POST /api/auth/login HTTP/1.1" 401 Unauthorized
```

Hình 4.25. Đo thời gian phản hồi API xác thực người dùng thất bại

Đo thời gian phản hồi API xác thực người dùng thành công:



```
phamduong@phamduong-Vostro-5502: ~  
phamduong@phamduong-Vostro-5502: ~$ curl -w "Response time: %{time_total}s\n" -o /dev/null -s -X POST -H "Content-Type: application/json" -d '{"username": "duong", "password": "duong123"}' http://localhost:4000/api/auth/login  
Response time: 0.230902s  
phamduong@phamduong-Vostro-5502: ~$  
  
(venv) phamduong@phamduong-Vostro-5502: ~/DATN/centerBE$ python main.py  
INFO: Started server process [18401]  
INFO: Waiting for application startup.  
INFO: Application startup complete.  
INFO: Uvicorn running on http://0.0.0.0:4000 (Press CTRL+C to quit)  
INFO: 127.0.0.1:37456 - "POST /api/auth/Login HTTP/1.1" 401 Unauthorized  
INFO: 127.0.0.1:58732 - "POST /api/auth/Login HTTP/1.1" 200 OK
```

Hình 4.26. Đo thời gian phản hồi API xác thực người dùng thành công

Từ đó có thể thấy rằng hành động xác thực người dùng sẽ mất khoảng 0,23s cho cả trường hợp thông tin xác thực failed hoặc success. Tất cả API khác cũng đo thời gian phản hồi bằng công cụ cURL đều cho ra lượng thời gian phản hồi khá là nhanh, phù hợp với hiệu suất của hệ thống.

- Đo CPU và RAM usage trên server. (top, htop)
- Ở trang Sensor Details, khi ấn ‘Start scan‘ quá lâu, trong bối cảnh nhiều thiết bị tại không gian quét thì tại bảng Scanresult sẽ có số lượng entry rất lớn. Khi bảng ‘Scanresult‘ có số lượng entry rất lớn, việc export CSV sẽ làm UI bị treo do quá trình xử lý đồng bộ blocking main thread. Vấn đề ở đây là bởi vì cả server sẽ bị Blocking I/O để đợi cho việc export CSV hoàn thành khi ấn ‘Stop scan‘.

#### 4.3.3 Đánh giá kết quả thực tế so với thiết kế

Kết quả thử nghiệm cho thấy hệ thống đáp ứng tốt các chức năng quản lý, giám sát thiết bị và thu thập dữ liệu wifi. Dữ liệu được truyền tải ổn định, hiển thị trực quan trên giao diện web. Hệ thống dễ dàng mở rộng số lượng thiết bị và đáp ứng nhu cầu truy xuất dữ liệu lớn.

#### 4.3.4 Vấn đề phát sinh và cách khắc phục

Trong quá trình triển khai, một số vấn đề đã được phát hiện như:

- Dữ liệu thô được chuyển tiếp giữa các module có thể dễ dàng bị đánh cắp.
- Một số lỗi phần mềm nhỏ liên quan đến định dạng dữ liệu, xác thực hoặc giao tiếp giữa các module (ThingsBoard, Backend, Frontend).
- Thông tin người dùng được chuyển tiếp giữa các module dễ bị theo dõi.

Những vấn đề trong quá trình triển khai đã có tồn đọng song đã tìm ra các giải pháp lần lượt tương ứng. Các giải pháp khắc phục bao gồm:

- Mã hóa đường truyền giữa Thingsboard và Backend bằng SSL/TLS.
- Bổ sung kiểm tra lỗi, cập nhật phần mềm định kỳ, tăng cường bảo mật và kiểm soát truy cập hệ thống.
- Mã hóa mật khẩu người dùng bằng Bcrypt (mã hóa 1 chiều)

#### 4.4 Thiết bị dò quét

##### 4.4.1 Quy trình triển khai

Quá trình triển khai hệ thống được thực hiện qua các bước chính sau:

- **Lắp ráp và sản xuất:** Các module phần cứng được lắp ráp theo thiết kế kỹ thuật, kiểm tra sơ bộ trước khi lập trình.
- **Lập trình:** Viết và nạp firmware cho các node cảm biến và gateway, phát triển phần mềm backend và giao diện điều khiển.
- **Triển khai thực tế:** Lắp đặt thiết bị tại vị trí thực tế, kết nối mạng và cấu hình hệ thống.

##### 4.4.2 Thử nghiệm và đo kiểm

Các hoạt động thử nghiệm được tiến hành nhằm kiểm tra:

- **Thử nghiệm chức năng:** Đánh giá các tính năng chính của hệ thống hoạt động đúng theo yêu cầu.
- **Kiểm tra độ bền:** Thử nghiệm các thiết bị dưới các điều kiện môi trường khác nhau để đảm bảo độ ổn định.
- **Hiệu suất:** Đo đặc tốc độ truyền dữ liệu, độ trễ, và khả năng xử lý đồng thời.

##### 4.4.3 Đánh giá kết quả thực tế so với thiết kế

Kết quả thử nghiệm được so sánh với các mục tiêu đặt ra ban đầu, đánh giá mức độ phù hợp, ưu điểm cũng như những điểm còn chưa đạt yêu cầu.

##### 4.4.4 Vấn đề phát sinh và cách khắc phục

Trong quá trình triển khai, một số vấn đề đã được phát hiện như:

- Lỗi kết nối mạng không ổn định do yếu tố môi trường.
- Thiết bị gặp hiện tượng quá tải khi xử lý lượng dữ liệu lớn.
- Các lỗi phần mềm nhỏ liên quan đến giao tiếp giữa các module.

Các giải pháp xử lý bao gồm:

- Tối ưu cấu hình mạng và sử dụng bộ khuếch đại tín hiệu.
- Tối ưu thuật toán xử lý dữ liệu và phân chia tải.
- Cập nhật và sửa lỗi phần mềm kịp thời.

### **Kết luận chương**

Chương 4 chúng em đã mô tả chi tiết quá trình triển khai và thử nghiệm hệ thống, đồng thời đánh giá kết quả đạt được và các khó khăn gặp phải trong thực tế. Hệ thống đã hoạt động cơ bản ổn định và đạt gần như đầy đủ các yêu cầu đề ra, tuy nhiên vẫn còn một số điểm cần cải tiến để nâng cao độ tin cậy và hiệu suất.

## CHƯƠNG 5. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

### Mở đầu chương

Chương 5 tổng kết toàn bộ quá trình thiết kế và phát triển hệ thống thiết bị dò quét Wi-Fi. Trong chương này, chúng em đánh giá các kết quả đạt được so với mục tiêu ban đầu, phân tích những hạn chế còn tồn tại và đề xuất định hướng phát triển trong tương lai nhằm hoàn thiện hệ thống.

### 5.5 Tóm tắt kết quả đạt được

Trong quá trình thiết kế và triển khai, đối với thiết bị dò quét Wi-Fi, chúng em đã xây dựng thành công kiến trúc thiết bị di động với khả năng hoạt động độc lập, tích hợp card Wi-Fi Intel AX200/AX210 để thu thập dữ liệu CSI. Thiết bị thực hiện tốt các chức năng quét và hiển thị Access Point, kể cả AP ẩn và các Station, đồng thời hỗ trợ sắp xếp, lọc kết quả theo kênh tần số. Các chức năng nhập/xuất dữ liệu định dạng Excel, lưu trữ dữ liệu cục bộ, quản lý Wi-Fi card, cấu hình thủ công, nhận diện nhà sản xuất thiết bị qua OUI, thu thập và hiển thị CSI, lưu trữ dữ liệu CSI, đồng bộ với hệ thống trung tâm và khóa thao tác khi kết nối đều đã được triển khai hiệu quả.

Về phía phần mềm quản lý trung tâm, hệ thống được thiết kế với kiến trúc quản lý tập trung, có khả năng quản lý nhiều thiết bị cùng lúc. API với đầy đủ các endpoint phục vụ quản lý sensor, card mạng và cơ sở dữ liệu đã được xây dựng hoàn chỉnh. Hệ thống hỗ trợ giao tiếp thời gian thực qua WebSocket, có module xác thực và quản lý người dùng, cung cấp đầy đủ các chức năng CRUD cho sensor và thiết bị, cho phép quản lý và cấu hình card mạng từ xa, điều khiển quá trình quét từ xa, truy vấn danh sách AP, Station, lịch sử hoạt động và mối quan hệ giữa các thiết bị trong mạng. Ngoài ra, phần mềm còn triển khai hệ thống thông báo, cảnh báo real-time, đồng bộ dữ liệu từ các thiết bị, giao diện quản lý trực quan và cơ chế phân quyền, bảo mật truy cập đáp ứng các yêu cầu thực tế.

### 5.6 Những hạn chế còn tồn đọng

Bên cạnh các kết quả đạt được, hệ thống vẫn còn tồn tại một số hạn chế cần tiếp tục hoàn thiện trong các giai đoạn phát triển tiếp theo. Hiệu suất xử lý và phân tích dữ liệu CSI với dung lượng lớn trong thời gian thực còn gặp khó khăn, đặc biệt khi số lượng thiết bị hoặc card Wi-Fi tăng lên, ảnh hưởng đến khả năng mở rộng của hệ thống. Việc hỗ trợ phần cứng hiện tại mới chỉ giới hạn ở một số loại card Wi-Fi nhất định, chưa tương thích rộng rãi với các dòng card khác, điều này làm hạn chế khả năng ứng dụng hệ thống trong các môi trường thiết bị đa dạng. Các tính năng phân tích, thống kê, báo cáo và cảnh báo còn ở mức cơ bản, chưa khai thác hết tiềm năng của dữ liệu CSI. Cơ chế bảo mật, xác thực và phân quyền vẫn cần được hoàn thiện để đáp ứng các yêu cầu triển khai thực tế quy mô lớn, đặc biệt trong môi trường doanh nghiệp. Ngoài ra, khả

năng tích hợp với các nền tảng, hệ thống giám sát mạng bên ngoài còn hạn chế, làm giảm tính linh hoạt và khả năng mở rộng của hệ thống.

### **5.7 Định hướng phát triển trong tương lai**

Để nâng cao hiệu quả và mở rộng phạm vi ứng dụng của hệ thống, chúng em đưa ra những định hướng để phát triển, đó là tập trung tối ưu hóa hiệu suất, cải tiến thuật toán xử lý dữ liệu CSI nhằm tăng tốc độ quét, phân tích và giảm độ trễ khi thu thập dữ liệu thời gian thực. Hệ thống sẽ được mở rộng hỗ trợ thêm nhiều loại card Wi-Fi khác nhau, đồng thời cập nhật các chuẩn Wi-Fi mới như Wi-Fi 6E, Wi-Fi 7 để tăng tính tương thích. Các tính năng phân tích nâng cao sẽ được phát triển, ứng dụng các phương pháp AI/ML để phát hiện bất thường, phân tích hành vi, định vị trong nhà dựa trên dữ liệu CSI. Giao diện người dùng sẽ được nâng cấp theo hướng trực quan, bổ sung các dashboard, biểu đồ, báo cáo và chức năng cảnh báo thông minh. Về bảo mật, hệ thống sẽ áp dụng các phương pháp xác thực nâng cao, mã hóa dữ liệu truyền tải và lưu trữ, phân quyền chi tiết cho từng nhóm người dùng. Ngoài ra, khả năng tích hợp sẽ được mở rộng thông qua việc xây dựng API mở, hỗ trợ kết nối với các hệ thống quản lý mạng, IoT hoặc các nền tảng phân tích dữ liệu khác. Hệ thống cũng sẽ hướng tới các ứng dụng mới như nhận diện chuyển động, giám sát an ninh, chăm sóc sức khỏe thông minh dựa trên dữ liệu CSI.

### **Kết luận chương**

Chương 5 đã tổng kết quá trình thiết kế và phát triển hệ thống thiết bị dò quét Wi-Fi cùng phần mềm quản lý trung tâm, đồng thời chỉ ra các kết quả nổi bật đã đạt được cũng như những hạn chế còn tồn tại. Hệ thống đã đáp ứng được các yêu cầu chức năng đề ra ban đầu, từ việc thu thập, phân tích dữ liệu CSI, quản lý tập trung đến giao tiếp thời gian thực giữa các thiết bị. Tuy nhiên, để hệ thống thực sự hoàn thiện và có khả năng ứng dụng rộng rãi trong thực tế, cần tiếp tục tối ưu, mở rộng cả về phần cứng lẫn phần mềm theo các định hướng đã đề xuất. Những kết quả đạt được là nền tảng vững chắc để hệ thống tiếp tục phát triển, mở ra nhiều cơ hội ứng dụng công nghệ CSI trong các lĩnh vực khác nhau của công nghệ thông tin và truyền thông hiện đại.

## KẾT LUẬN

## TÀI LIỆU THAM KHẢO

- [1] Viblo, “Http request và http response,” 4 2020, truy cập ngày 16/06/2025. [Online]. Available: <https://viblo.asia/p/http-request-va-http-response-naQZR42G5vx>
- [2] I. Fette and A. Melnikov, “The websocket protocol,” Internet Engineering Task Force (IETF), RFC 6455, December 2011, request for Comments. [Online]. Available: <https://datatracker.ietf.org/doc/html/rfc6455>
- [3] *Information technology — Message Queuing Telemetry Transport (MQTT) v3.1.1*, International Organization for Standardization Std. ISO/IEC 20922:2016, 2016. [Online]. Available: <https://www.iso.org/standard/69466.html>
- [4] A. Banks and R. Gupta, “Mqtt version 3.1.1,” OASIS, OASIS Standard, October 2014, oASIS Standard. [Online]. Available: <https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html>
- [5] Wikipedia, “Single-page application,” 2024, bài viết được cập nhật thường xuyên, truy cập ngày 16/06/2025. [Online]. Available: [https://en.wikipedia.org/wiki/Single-page\\_application](https://en.wikipedia.org/wiki/Single-page_application)
- [6] Heavy.AI, “What is server-side rendering? definition and faqs,” 2024, truy cập ngày 16/06/2025. [Online]. Available: <https://www.heavy.ai/technical-glossary/server-side-rendering>
- [7] I. Bagchi, “Building the web of tomorrow: Ssr vs. csr - which path should you choose?” 2023, truy cập ngày 16/06/2025. [Online]. Available: <https://dev.to/ishanbagchi/building-the-web-of-tomorrow-ssr-vs-csr-which-path-should-you-choose-19bb>
- [8] ThingsBoard, “Thingsboard documentation,” 2025, tài liệu kỹ thuật chính thức, truy cập ngày 16/06/2025. [Online]. Available: <https://thingsboard.io/docs/>
- [9] T. P. G. D. Group, *PostgreSQL Documentation*, 2025, truy cập ngày 16/06/2025. [Online]. Available: <https://www.postgresql.org/docs/>
- [10] DigitalOcean, “An introduction to postgresql,” 2024, truy cập ngày 16/06/2025. [Online]. Available: <https://www.digitalocean.com/community/tutorials/an-introduction-to-postgresql>
- [11] S. Ramirez, “Fastapi - modern, fast (high-performance), web framework for building apis with python,” 2025, tài liệu chính thức, truy cập ngày 16/06/2025. [Online]. Available: <https://fastapi.tiangolo.com/>



- [12] —, “Fastapi tutorial - user guide,” 2025, hướng dẫn sử dụng, truy cập ngày 16/06/2025. [Online]. Available: <https://fastapi.tiangolo.com/tutorial/>
- [13] Vercel, “Next.js - the react framework for the web,” 2025, tài liệu chính thức, truy cập ngày 16/06/2025. [Online]. Available: <https://nextjs.org/>
- [14] —, “Next.js - the react framework,” 2025, mã nguồn mở trên GitHub, truy cập ngày 16/06/2025. [Online]. Available: <https://github.com/vercel/next.js>
- [15] ThingsBoard Team. (2024) Installing thingsboard ce on ubuntu server. [Online]. Available: <https://thingsboard.io/docs/user-guide/install/ubuntu/>

## **PHỤ LỤC**

### **A Một số phương pháp đo và hiệu chuẩn**

Phụ lục cần thêm (nếu có) ...